Stream: Internet Engineering Task Force (IETF)
RFC: 10021
Category: Standards Track
Published: November 2026
ISSN: 2070-1721
Authors: M. Tiloca, *RISE AB*; G. Selander, *Ericsson AB*; F. Palombini, *Ericsson AB*; J. Preuß Mattsson, *Ericsson AB*; R. Höglund, *RISE AB*

RFC 10021
Group Object Security for Constrained RESTful Environments (Group OSCORE)

Abstract

This document defines the security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE), providing end-to-end security of messages exchanged with the Constrained Application Protocol (CoAP) between members of a group, e.g., sent over IP multicast. In particular, Group OSCORE provides protection and source authentication for group requests sent from a client to multiple servers and for the corresponding responses. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive symmetric pairwise keys to use with another member of the group for pairwise-protected communication. Group OSCORE can be used between endpoints communicating with CoAP or CoAP-mappable HTTP.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc10021.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 5

1.1. Terminology 7

2. Security Context 9

2.1. Common Context 11

2.1.1. AEAD Algorithm 11

2.1.2. HKDF Algorithm 11

2.1.3. ID Context 11

2.1.4. Common IV 11

2.1.5. Authentication Credential Format 11

2.1.6. Group Manager Authentication Credential 11

2.1.7. Group Encryption Algorithm 12

2.1.8. Signature Algorithm 12

2.1.9. Signature Encryption Key 13

2.1.10. Pairwise Key Agreement Algorithm 13

2.2. Sender Context and Recipient Context 14

2.3. Establishment of Security Context Parameters 14

2.4. Authentication Credentials 15

2.5. Pairwise Keys 16

2.5.1. Derivation of Pairwise Keys 17

2.5.2. ECDH with Montgomery Coordinates 18

2.5.3. Usage of Sequence Numbers 19

2.5.4. Security Context for Pairwise Mode 19

2.6. Update of Security Context 20

2.6.1. Loss of the Varying Part of the Security Context 20

2.6.2. Exhaustion of Sender Sequence Number Space 21
2.6.3. Retrieving New Security Context Parameters 22
3. The COSE Object 24
3.1. Countersignature 24
3.1.1. Clarifications on Using a Countersignature 24
3.2. The 'kid' and 'kid context' Parameters 25
3.3. Nonce Computation 25
3.4. Additional Authenticated Data 25
4. OSCORE Header Compression 27
4.1. Encoding of the OSCORE Option Value and Group OSCORE Payload 28
4.2. Keystream Derivation for Countersignature Encryption/Decryption 28
4.3. Examples of Compressed COSE Objects 29
4.3.1. Examples in Group Mode 29
4.3.2. Examples in Pairwise Mode 31
5. Message Binding, Sequence Numbers, Freshness, and Replay Protection 32
5.1. Supporting Multiple Responses in Long Exchanges 32
5.2. Freshness 32
5.3. Replay Protection 33
5.3.1. Replay Protection of Responses 33
6. Message Reception 34
7. Message Processing in Group Mode 35
7.1. Protecting the Request 37
7.2. Verifying the Request 38
7.3. Protecting the Response 39
7.4. Verifying the Response 41
7.5. External Signature Checkers 44
8. Message Processing in Pairwise Mode 45
8.1. Preconditions 46
8.2. Main Differences from OSCORE 46
8.3. Protecting the Request 47

8.4. Verifying the Request 47
8.5. Protecting the Response 48
8.6. Verifying the Response 48
9. Challenge-Response-Based Freshness and Replay Window Recovery 49
10. Implementation Compliance 51
11. Web Linking 52
12. The Group Manager 53
12.1. Setup of New Endpoints 54
12.2. Management of Group Keying Material 54
12.2.1. Recycling of Identifiers 57
12.3. Support for Signature Checkers 60
13. Security Considerations 60
13.1. Security of the Group Mode 61
13.1.1. Example of Need for Proof of Group Membership 63
13.2. Security of the Pairwise Mode 63
13.3. Uniqueness of (key, nonce) 64
13.4. Management of Group Keying Material 64
13.4.1. Denial of Service 64
13.5. Update of Security Context and Key Rotation 65
13.5.1. Late Update on the Sender 65
13.5.2. Late Update on the Recipient 66
13.6. Collision of Group Identifiers 66
13.7. Cross-Group Message Injection 66
13.7.1. Attack Description 67
13.7.2. Attack Prevention in Group Mode 68
13.8. Prevention of Group Cloning Attack 68
13.9. Group OSCORE for Unicast Requests 68
13.10. End-to-End Protection 70
13.11. Master Secret 70
13.12. Replay Protection 70

13.13. Message Ordering 71
13.14. Message Freshness 71
13.15. Client Aliveness 72
13.16. Cryptographic Considerations 72
13.17. Message Segmentation 73
13.18. Privacy Considerations 74
14. IANA Considerations 75
14.1. OSCORE Flag Bits Registry 75
14.2. Target Attributes Registry 75
15. References 75
15.1. Normative References 75
15.2. Informative References 77
Appendix A. Assumptions and Security Objectives 80
A.1. Assumptions 80
A.2. Security Objectives 81
Appendix B. List of Use Cases 82
Appendix C. Example of Group Identifier Format 84
Appendix D. Responsibilities of the Group Manager 85
Acknowledgements 86
Authors' Addresses 86

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol specifically designed for constrained devices and networks [RFC7228]. Group communication for CoAP [RFC10020] addresses use cases where deployed devices benefit from a group communication model, for example, to reduce latencies, improve performance, and reduce bandwidth utilization. Use cases include lighting control, integrated building control, software and firmware updates, parameter and configuration updates, commissioning of constrained networks, and emergency multicast (see Appendix B). Group communication for CoAP [RFC10020] mainly uses UDP/IP multicast as the underlying data transport.

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] is a security protocol that protects CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC9052] [RFC9053] and provides end-to-end encryption, integrity, replay protection, and binding of response to request between a sender and a recipient, independent of the transport layer also in the presence of intermediaries. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields into a COSE object, which is conveyed within the CoAP payload and the CoAP OSCORE Option of the protected message, thereby replacing those message fields with an authenticated and encrypted object.

This document defines Group OSCORE, a security protocol for group communication with CoAP [RFC10020] and for CoAP-mappable HTTP requests and responses, providing the same end-to-end security properties as OSCORE also in the case where requests have multiple recipients. In particular, Group OSCORE provides protection and source authentication for group requests sent from a client to multiple servers and for the corresponding responses. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive symmetric pairwise keys to use with another member of the group for pairwise-protected communication. Just like OSCORE, Group OSCORE is independent of the transport layer and works wherever CoAP does.

As with OSCORE, it is possible to combine Group OSCORE with communication security on other layers. One example is the use of transport layer security, such as DTLS [RFC9147], between one client and one proxy, or between one proxy and one server. This prevents observers from accessing addressing information conveyed in CoAP options that would not be protected by Group OSCORE, but would be protected by DTLS. These options include Uri-Host, Uri-Port, and Proxy-Uri. Note that DTLS does not define how to secure messages sent over IP multicast and cannot be used for end-to-end protection over a proxy. Group OSCORE is also intended to work with OSCORE-capable proxies [OSCORE-PROXIES], thereby enabling, for example, nested OSCORE operations with OSCORE-protected communication between a CoAP client and a proxy, carrying messages that are additionally protected with Group OSCORE between the CoAP client and the target CoAP servers.

Group OSCORE defines two modes of operation that can be used independently or together:

- In the group mode, Group OSCORE requests and responses are digitally signed with the private key of the sender and the signature is embedded in the protected CoAP message. The group mode supports all COSE signature algorithms as well as signature verification by intermediaries. This mode is defined in Section 7.
- In the pairwise mode, two group members exchange OSCORE requests and responses (typically) over unicast, and the messages are protected with symmetric keys not known by the other group members. These symmetric keys are derived from Diffie-Hellman shared secrets, which are calculated with the asymmetric keys of the sender and recipient, allowing for shorter integrity tags and therefore lower message overhead. This mode is defined in Section 8.

Both modes provide source authentication of CoAP messages. The application decides what mode to use, potentially on a per-message basis. Such decisions can be based, for instance, on preconfigured policies or dynamic assessing of the target recipient and/or resource, among other

things. One important case is when requests are protected in group mode and responses are protected in pairwise mode. Since such responses convey shorter integrity tags instead of bigger, full-fledged signatures, this significantly reduces the message overhead in the case of many responses to one request.

A special deployment of Group OSCORE consists in using the pairwise mode only. For example, consider the case of a constrained-node network [RFC7228] with a large number of CoAP endpoints and the objective to establish secure communication between any pair of endpoints with a small provisioning effort and message overhead. Since the total number of security associations that needs to be established grows with the square of the number of endpoints, it is desirable to restrict the amount of secret keying material provided to each endpoint. Moreover, a key establishment protocol would need to be executed for each security association. One solution to this issue is to deploy Group OSCORE, with the endpoints being part of a group, and to use the pairwise mode. This solution offers the benefit of providing a single shared secret while distributing only the public keys of group members or a subset of those. After that, a CoAP endpoint can locally derive the keying material to use with the other endpoint in the group and protect CoAP communications with very low overhead [CoAP-SEC-PROT].

In some circumstances, Group OSCORE messages may be transported in HTTP, e.g., when they are protected with the pairwise mode and target a single recipient, or when they are protected with the group mode and target multiple CoAP recipients through cross-protocol translators such as HTTP-to-CoAP proxies [RFC8075] [CoAP-GROUP-PROXY]. The use of Group OSCORE with HTTP is as defined for OSCORE in Section 11 of [RFC8613].

1.1. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], including "endpoint", "client", "server", "sender", and "recipient"; group communication for CoAP [RFC10020]; Observe [RFC7641]; Concise Binary Object Representation (CBOR) [RFC8949]; Concise Data Definition Language (CDDL) [RFC8610]; COSE [RFC9052] [RFC9053]; and related countersignatures [RFC9338].

Readers are also expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE, such as "Security Context" and "Master Secret", defined in [RFC8613].

Terminology for constrained environments, such as "constrained device" and "constrained-node network", is defined in [RFC7228].

This document also refers to the following terminology.

Keying material: Data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].

Authentication credential: Information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of formats of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC5280], and C509 certificates [C509-CERT]. Further details about authentication credentials are provided in Section 2.4.

Group: A set of endpoints that share group keying material and security parameters (Common Context; see Section 2). That is, unless otherwise specified, the term "group" used in this document refers to a "security group" (see Section 2.1 of [RFC10020]), not to be confused with "CoAP group" or "application group".

Group Manager: An entity responsible for a group. A Group Manager is not required to be an actual group member or to participate in group communication. The operations of the Group Manager are defined in Section 12, and its responsibilities are listed in Appendix D.

Silent server: A member of a group that performs only group mode processing on incoming requests and never sends responses protected with Group OSCORE. For CoAP group communications, requests are normally sent without necessarily expecting a response. A silent server may send unprotected responses as error responses to report a Group OSCORE error.

Group Identifier (Gid): An identifier that is assigned to the group and is unique within the set of groups of a given Group Manager. The Gid value changes every time the group is rekeyed (see Section 12.2).

Birth Gid: With respect to a group member, the Gid obtained by that group member upon (re-)joining the group.

Key Generation Number: An integer value identifying the current version of the keying material used in a group.

Source authentication: Evidence that a received message in the group originated from a specific identified group member. This also provides assurance that the message was not tampered with by anyone, be it a different legitimate group member or an endpoint that is not a group member.

Group request: A CoAP request message sent by a client in the group to servers in that group.

Long exchange: An exchange of messages associated with a request that is a group request and/or an Observe request [RFC7641].

In either case, multiple responses can follow from the same server to the request associated with the long exchange, even if the request is not an Observe request (see Section 3.1.6 of [RFC10020]). The client terminates a long exchange when freeing up the CoAP Token value used for the associated request, for which no further responses will be accepted afterwards.

2. Security Context

This document refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol; see Section 1.1. All members of a group maintain a Security Context as defined in this section.

How the Security Context is established by the group members is out of scope for this document, but if there is more than one Security Context applicable to a message, then the endpoints **MUST** be able to determine which Security Context was latest established. The management of information about the group (i.e., identifiers, OSCORE input parameters, and keying material) is described in terms of a Group Manager (see Section 12).

An endpoint of the group may use the group mode (see Section 7), the pairwise mode (see Section 8), or both, depending on the modes it supports and the parameters of the Security Context. The Security Context of Group OSCORE extends the OSCORE Security Context defined in Section 3 of [RFC8613] as follows (see Figure 1).

- One Common Context shared by all the endpoints in the group and extended as defined below.
  - The new parameter Authentication Credential Format (see Section 2.1.5), specifying the format of authentication credentials used in the group (see Section 2.4).
  - The new parameter Group Manager Authentication Credential, specifying the authentication credential of the Group Manager responsible for the group (see Section 2.1.6).
  - For the group mode, the Common Context is extended with the following new parameters.
    - Group Encryption Algorithm, specifying the algorithm used for encrypting and decrypting messages protected in group mode (see Section 2.1.7).
    - Signature Algorithm, specifying the algorithm used for computing and verifying the countersignature of messages protected in group mode (see Section 2.1.8).
    - Signature Encryption Key, specifying the symmetric key used for deriving a keystream, which, in turn, is used for encrypting and decrypting the countersignature of messages protected in group mode (see Section 2.1.9).
  - For the pairwise mode, the Common Context is extended with a Pairwise Key Agreement Algorithm (see Section 2.1.10) used for the agreement on a static-static Diffie-Hellman shared secret from which pairwise keys are derived (see Section 2.5.1).

  The content of the Common Context is long-term, as it is meant to be stable once the Common Context is established.
- One Sender Context extended with the following new parameters.
  - The endpoint's own private key used to sign messages protected in group mode (see Section 7) or for deriving pairwise keys used with the pairwise mode (see Section 2.5).
  - The endpoint's own authentication credential containing its public key (see Section 2.4).

- For the pairwise mode, the Sender Context is extended with the Pairwise Sender Keys associated with the other endpoints (see Section 2.5).

Except for the Sender Sequence Number defined in Section 3.1 of [RFC8613], the content of the Sender Context is long-term, as it is meant to be stable once the Sender Context is established.

If the endpoint is configured exclusively as a silent server (see Section 1.1), then the Sender Context is omitted.

- One Recipient Context for each other endpoint from which messages are received. It is not necessary to maintain Recipient Contexts associated with endpoints from which messages are not (expected to be) received.
  - Each Recipient Context is extended with the authentication credential of the other endpoint, which is used to verify the signature of messages protected in group mode or to derive the pairwise keys used with the pairwise mode (see Section 2.5).
  - For the pairwise mode, each Recipient Context is extended with the Pairwise Recipient Key associated with the other endpoint (see Section 2.5).

  Except for the Replay Window defined in Section 3.1 of [RFC8613], the content of each Recipient Context is long-term, as it is meant to be stable once the Recipient Context is established.

The varying part of the Group OSCORE Security Context is composed of the Sender Sequence Number in the Sender Context and the Replay Windows in the different Recipient Contexts.

```
+-------------------+-------------------------------------------------+
| Context Component | New Information Elements                        |
+-------------------+-------------------------------------------------+
| Common Context    |   Authentication Credential Format              |
|                   |   Group Manager Authentication Credential       |
|                   | * Group Encryption Algorithm                    |
|                   | * Signature Algorithm                           |
|                   | * Signature Encryption Key                      |
|                   | ^ Pairwise Key Agreement Algorithm              |
+-------------------+-------------------------------------------------+
| Sender Context    |   Endpoint's own private key                    |
|                   |   Endpoint's own authentication credential      |
|                   | ^ Pairwise Sender Keys for the other endpoints  |
+-------------------+-------------------------------------------------+
| Each              |   Other endpoint's authentication credential    |
| Recipient Context | ^ Pairwise Recipient Key for the other endpoint |
+-------------------+-------------------------------------------------+
```

Figure 1: Additions to the OSCORE Security Context

Note: The elements labeled with * and with ^ are relevant only for the group mode and only for the pairwise mode, respectively.

2.1. Common Context

The following sections specify how the Common Context is used and extended compared to [RFC8613]. The Common Context may be acquired from the Group Manager (see Section 12).

2.1.1. AEAD Algorithm

The Authenticated Encryption with Associated Data (AEAD) Algorithm (see Section 3.1 of [RFC8613]) identifies the COSE AEAD algorithm to use for encryption and decryption when messages are protected using the pairwise mode (see Section 8). This algorithm **MUST** provide integrity protection. If this parameter is not set, the pairwise mode is not used in the group.

2.1.2. HKDF Algorithm

The HMAC-based Key Derivation Function (HKDF) Algorithm (see Section 3.1 of [RFC8613]) identifies the used key derivation function, which **MUST** be one of the HKDF [RFC5869] algorithms defined for COSE to use HMAC as the underlying PRF (see Section 5.1 of [RFC9053]) and registered at [COSE.Algorithms].

2.1.3. ID Context

The ID Context parameter (see Sections 3.1 and 3.3 of [RFC8613]) contains the Gid of the group. The choice of the Gid format is application-specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle potential collisions between Gids (see Section 13.6).

2.1.4. Common IV

The Common IV parameter (see Section 3.1 of [RFC8613]) identifies the Common IV used in the group. Differently from OSCORE, the length of the Common IV is determined as follows:

- If only one among the AEAD Algorithm and the Group Encryption Algorithm is set (see Sections 2.1.1 and 2.1.7), the length of the Common IV is the nonce length for the set algorithm.
- If both the AEAD Algorithm and the Group Encryption Algorithm are set, the length of the Common IV is the greatest nonce length among those of the two algorithms.

If the Group Encryption Algorithm is A128CTR, A192CTR, or A256CTR (see Section 4 of [RFC9459]), then the length of the nonce used by that algorithm is 12 bytes (see Section 2.1.7).

2.1.5. Authentication Credential Format

The new parameter Authentication Credential Format specifies the format of authentication credentials used in the group.

2.1.6. Group Manager Authentication Credential

The new parameter Group Manager Authentication Credential specifies the authentication credential of the Group Manager, including the Group Manager's public key. The endpoint **MUST** achieve proof of possession of the corresponding private key. As an example, such proof of

possession is possible to achieve during the join process provided by the realization of Group Manager specified in [ACE-KEY-GROUP-OSCORE]. Further details on the provisioning of the Group Manager's authentication credential to the group members are out of the scope of this document.

2.1.7. Group Encryption Algorithm

The new parameter Group Encryption Algorithm identifies the algorithm to use for encryption and decryption when messages are protected in group mode (see Section 7). This algorithm **MAY** provide integrity protection. If it does not, integrity protection is still provided by the countersignature added to the message due to the use of the group mode. If this parameter is not set, the group mode is not used in the group.

In order to be eligible to use as Group Encryption Algorithm, a non-authenticated algorithm **MUST** ensure that the same key is not reused with the same IV or intermediate values used in the algorithm, e.g., for algorithms that increment the IV internally. If a non-authenticated algorithm does not fulfill the requirement above, that algorithm **MUST NOT** be used as Group Encryption Algorithm.

Examples of non-authenticated algorithms that can be used as Group Encryption Algorithm are A128CTR, A192CTR, and A256CTR (see Section 4 of [RFC9459]). When either of those three algorithms is used, the following applies:

- A 12-byte nonce **MUST** be computed as defined in Section 3.3.
- The Initialization Vector (IV) used in Section 4 of [RFC9459] is equivalent to the nonce above (12 bytes) concatenated with 0x00000000 (4 bytes) in this order.
- The algorithm **MUST NOT** be used to encrypt a plaintext or decrypt a ciphertext whose length is larger than 64 GB (i.e., 2^{36} bytes).

The non-authenticated algorithms A128CBC, A192CBC, and A256CBC (see Section 5 of [RFC9459]) **MUST NOT** be used as Group Encryption Algorithm.

Future specifications can admit alternative non-authenticated algorithms that can be used as Group Encryption Algorithm. When doing so, it **MUST** be defined how to securely compose the IV and possible intermediate values used in the algorithm, building on the nonce computed as defined in Section 3.3. Absent such a specification, alternative non-authenticated algorithms **MUST NOT** be used as Group Encryption Algorithm.

2.1.8. Signature Algorithm

The new parameter Signature Algorithm identifies the digital signature algorithm used for computing and verifying the countersignature on the COSE object (see Sections 3.2 and 3.3 of [RFC9338]) when messages are protected in group mode (see Section 7). If this parameter is not set, the group mode is not used in the group.

2.1.9. Signature Encryption Key

The new parameter Signature Encryption Key specifies the symmetric key for deriving a keystream to encrypt/decrypt a countersignature (see Section 4.2) when a message is protected in group mode (see Section 7).

The Signature Encryption Key is derived as defined for Sender Keys and Recipient Keys in Section 3.2.1 of [RFC8613] with the following differences:

- The 'id' element of the 'info' array is the empty byte string.
- The 'alg_aead' element of the 'info' array specifies the Group Encryption Algorithm from the Common Context (see Section 2.1.7), encoded as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" registry for this algorithm.
- The 'type' element of the 'info' array is "SEKey". The label is an ASCII string and does not include a trailing NUL byte.
- L and the 'L' element of the 'info' array are the size of the key for the Group Encryption Algorithm specified in the Common Context (see Section 2.1.7) in bytes. While the obtained Signature Encryption Key is never used with the Group Encryption Algorithm, its length was chosen to obtain a matching level of security.

2.1.10. Pairwise Key Agreement Algorithm

The new parameter Pairwise Key Agreement Algorithm identifies the Elliptic Curve Diffie-Hellman (ECDH) algorithm used to derive a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.5.1) to protect messages with the pairwise mode (see Section 8). If this parameter is not set, the pairwise mode is not used in the group.

When two endpoints compute their Diffie-Hellman shared secret, the Pairwise Key Agreement Algorithm takes the static-static Diffie-Hellman keys of the two endpoints as input. The lifetime of those keys is the same as the lifetime of the authentication credentials that the two endpoints use in the group. As detailed in Section 2.5.1, the derivation of the pairwise keys takes not only the Diffie-Hellman shared secret as input, but also group keying material from the latest established Security Context.

If the HKDF Algorithm specified in the Common Context is "HKDF SHA-256" (identified as "HMAC 256/256"), then the Pairwise Key Agreement Algorithm is "ECDH-SS + HKDF-256" (COSE algorithm encoding: -27).

If the HKDF Algorithm specified in the Common Context is "HKDF SHA-512" (identified as "HMAC 512/512"), then the Pairwise Key Agreement Algorithm is "ECDH-SS + HKDF-512" (COSE algorithm encoding: -28).

Note that the HKDF Algorithm in the Common Context is denoted by the corresponding COSE HMAC Algorithm. For example, the HKDF Algorithm "HKDF SHA-256" is specified as the HMAC Algorithm "HMAC 256/256".

More generally, if Pairwise Key Agreement Algorithm is set, it **MUST** identify a COSE algorithm such that i) it performs a direct ECDH Static-Static key agreement, and ii) it indicates the use of the same HKDF Algorithm used in the group as specified in the Common Context.

2.2. Sender Context and Recipient Context

The Sender ID **SHALL** be unique for each endpoint in a group with a certain triplet (Master Secret, Master Salt, Group Identifier); see Section 3.3 of [RFC8613].

The maximum length of a Sender ID in bytes equals L minus 6, where L is determined as follows.

- If only one among the AEAD Algorithm and the Group Encryption Algorithm is set (see Sections 2.1.1 and 2.1.7), then L is the nonce length for the set algorithm.
- If both the AEAD Algorithm and the Group Encryption Algorithm are set, then L is the smallest nonce length among those of the two algorithms.

With the exception of the authentication credential of the sender endpoint, a receiver endpoint can derive a complete Security Context from a received Group OSCORE message and the Common Context (see Section 2.3).

The authentication credentials in the Recipient Contexts and other information (e.g., the Sender IDs) associated with group members can be retrieved from the Group Manager upon joining the group (see Section 12). An authentication credential and other information associated with a group member can alternatively be acquired from the Group Manager at a later time, for example, the first time a message is received from a particular endpoint in the group (see Sections 7.2 and 7.4).

For severely constrained devices, it may be infeasible to simultaneously handle the ongoing processing of a recently received message in parallel with the retrieval of the sender endpoint's authentication credential. Such devices can be configured to drop a received message for which there is no (complete) Recipient Context and retrieve the sender endpoint's authentication credential in order to have it available to verify subsequent messages from that endpoint.

An endpoint may admit a maximum number of Recipient Contexts for the same Security Context, e.g., due to memory limitations. After reaching that limit, the endpoint has to delete a current Recipient Context to install a new one (see Section 2.6.1.2). It is up to the application to define the maximum number of Recipient Contexts for the same Security Context as well as policies for deleting Recipient Contexts.

2.3. Establishment of Security Context Parameters

OSCORE defines the derivation of Sender Context and Recipient Context (specifically, of Sender Keys and Recipient Keys) and of the Common IV from a set of input parameters (see Section 3.2 of [RFC8613]).

The derivation of Sender Keys, Recipient Keys, and Common IV defined in OSCORE applies also to Group OSCORE with the following modifications compared to Section 3.2.1 of [RFC8613]:

- If Group Encryption Algorithm in the Common Context is set (see Section 2.1.7), then the 'alg_aead' element of the 'info' array **MUST** specify Group Encryption Algorithm from the Common Context as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" registry for this algorithm.
- If Group Encryption Algorithm in the Common Context is not set, then the 'alg_aead' element of the 'info' array **MUST** specify AEAD Algorithm from the Common Context (see Section 2.1.1), as per Section 3.2.1 of [RFC8613].
- When deriving the Common IV, the 'L' element of the 'info' array **MUST** specify the length of the Common IV in bytes, which is determined as defined in Section 2.1.4.

2.4. Authentication Credentials

The authentication credentials of the endpoints in a group **MUST** be encoded according to the format used in the group as indicated by the Authentication Credential Format parameter in the Common Context (see Section 2.1.5). The authentication credential of the Group Manager **SHOULD** be encoded according to that same format in order to limit the number of formats that the group members have to support and handle, unless it is infeasible or impractical for the particular realization or instance of the Group Manager to have an own authentication credential encoded in that same format.

The format of authentication credentials **MUST** provide the public key and a comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable). If Group Encryption Algorithm in the Common Context is not set (see Section 2.1.7), then the public key algorithm is the Pairwise Key Agreement Algorithm used in the group (see Section 2.1.10); otherwise, the Signature Algorithm used in the group (see Section 2.1.8).

Examples of formats of authentication credentials are CWTs and CCSs [RFC8392], X.509 certificates [RFC5280], and C509 certificates [C509-CERT].

If the authentication credentials are X.509 certificates or C509 certificates, the public key algorithm is identified by the "algorithm" field of the "SubjectPublicKeyInfo" structure and by the "subjectPublicKeyAlgorithm" element, respectively.

If authentication credentials are CWTs or CCSs, then a COSE Key structure and its "kty" and "crv" parameters identify the types of pertinent public key algorithms. For example, the pair ("crv" = X25519, "kty" = OKP) indicates that the public key is meant to be used with X25519 ECDH key agreement; the pair ("crv" = Ed25519, "kty" = OKP) indicates that the public key is meant to be used with the Edwards-curve Digital Signature Algorithm (EdDSA); and the pair ("crv" = P-256, "kty" = EC2) indicates that the public key is meant to be used with the Elliptic Curve Digital Signature Algorithm (ECDSA) and/or with P-256 ECDH key agreement.

Authentication credentials are used to derive pairwise keys (see Section 2.5.1) and are included in the external Additional Authenticated Data (AAD) when processing messages (see Section 3.4). In both of these cases, an endpoint in a group **MUST** treat authentication credentials as opaque data, i.e., by considering the same binary representation made available to other endpoints in the group, possibly through a designated trusted source (e.g., the Group Manager).

For example, an X.509 certificate is provided as its direct binary serialization. If C509 certificates or CWTs are used as authentication credentials, each is provided as the binary serialization of a (possibly tagged) CBOR array. If CCSs are used as authentication credentials, each is provided as the binary serialization of a (possibly tagged) CBOR map.

If authentication credentials are CWTs or CCSs, then the untagged CWT or CCS associated with an entity is stored in the Security Context and used as authentication credential for that entity.

If authentication credentials are X.509/C509 certificates, CWTs, or CCSs and the authentication credential associated with an entity is provided within a chain or a bag, then only the end-entity certificate or end-entity untagged CWT/CCS is stored in the Security Context and used as authentication credential for that entity.

Storing whole authentication credentials rather than only a subset of those may result in a non-negligible storage overhead. On the other hand, it also ensures that authentication credentials are correctly used in a simple, flexible, and non-error-prone way, also taking into account future credential formats as entirely new or extending existing ones. In particular, it is ensured that:

- When used to derive pairwise keys and when included in the external AAD, authentication credentials can also specify possible metadata and parameters related to the included public key. Besides the public key algorithm, these comprise other relevant pieces of information such as key usage, expiration time, issuer, and subject.
- All endpoints using another endpoint's authentication credential use exactly the same binary serialization, as obtained and distributed by the credential provider (e.g., the Group Manager), and as originally crafted by the credential issuer. In turn, this does not require to define and maintain canonical subsets of authentication credentials and their corresponding encoding and spares endpoints from error-prone re-encoding operations.

Depending on the particular deployment and the intended group size, limiting the storage overhead of endpoints in a group can be an incentive for system/network administrators to prefer using a compact format of authentication credentials in the first place.

2.5. Pairwise Keys

In certain Elliptic Curve Cryptographic schemes, it is possible to use public/private key pairs with both signature operations (ECDSA or EdDSA) and key agreement operations (ECDH). This section specifies the derivation of "pairwise keys" for use in the pairwise mode defined in Section 8.

Group OSCORE keys used for both signature operations and key agreement operations **MUST** be used only for purposes related to Group OSCORE. These include the processing of messages with Group OSCORE, as well as performing proof of possession of private keys, e.g., upon joining a group through the Group Manager (see Section 12).

2.5.1. Derivation of Pairwise Keys

Using the Group OSCORE Security Context (see Section 2), a group member can derive AEAD keys to protect point-to-point communication between itself and each other endpoint X in the group by means of the AEAD Algorithm from the Common Context (see Section 2.1.1).

Analogous to the construction used by OSCORE in Section 3.2.1 of [RFC8613], the key derivation of these so-called pairwise keys relies on an HKDF Algorithm and is as defined below:

```
Pairwise Sender Key    = HKDF(Sender Key, IKM-Sender, info, L)
Pairwise Recipient Key = HKDF(Recipient Key, IKM-Recipient, info, L)

with

IKM-Sender    = Sender Auth Cred | Recipient Auth Cred | Shared Secret
IKM-Recipient = Recipient Auth Cred | Sender Auth Cred | Shared Secret
```

where:

- The Pairwise Sender Key is the AEAD key for processing outgoing messages addressed to endpoint X.
- The Pairwise Recipient Key is the AEAD key for processing incoming messages from endpoint X.
- HKDF is the OSCORE HKDF Algorithm [RFC8613] from the Common Context.
- The Sender Key from the Sender Context is used as salt in the HKDF when deriving the Pairwise Sender Key.
- The Recipient Key from the Recipient Context associated with endpoint X is used as salt in the HKDF, when deriving the Pairwise Recipient Key.
- Sender Auth Cred is the endpoint's own authentication credential from the Sender Context.
- Recipient Auth Cred is the endpoint X's authentication credential from the Recipient Context associated with endpoint X.
- The Shared Secret is computed as a cofactor Diffie-Hellman shared secret (see Section 5.7.1.2 of [NIST-800-56A]) using the Pairwise Key Agreement Algorithm. The endpoint uses its private key from the Sender Context and the other endpoint's public key included in Recipient Auth Cred. Note the requirement of validation of public keys in Section 13.16.

  If the other endpoint's public key has COSE Key Type "EC2" [RFC9053] (e.g., for the curves P-256, P-384, and P-521), then the public key is used as is. If the other endpoint's public key has COSE Key Type "OKP" [RFC9053], the procedure is described in Section 5 of [RFC7748]. In particular, if the public key is for X25519 or X448, it is used as is. Otherwise, if the public key is for the curve Ed25519 or Ed448, it is first mapped to Montgomery coordinates (see Section 2.5.2).

- IKM-Sender is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Sender Key. IKM-Sender is the byte string concatenation of Sender Auth Cred, Recipient Auth Cred, and the Shared Secret. The authentication credentials Sender Auth Cred and Recipient Auth Cred are binary-encoded as defined in Section 2.4.
- IKM-Recipient is the IKM used in the HKDF for the derivation of the Pairwise Recipient Key. IKM-Recipient is the byte string concatenation of Recipient Auth Cred, Sender Auth Cred, and the Shared Secret. The authentication credentials Recipient Auth Cred and Sender Auth Cred are binary-encoded as defined in Section 2.4.
- info and L are as defined in Section 3.2.1 of [RFC8613]. That is:
  - The 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).
  - L and the 'L' element of the 'info' array are the size of the key for the AEAD Algorithm from the Common Context (see Section 2.1.1) in bytes.

If EdDSA asymmetric keys are used, the Edward coordinates are mapped to Montgomery coordinates using the maps defined in Sections 4.1 and 4.2 of [RFC7748] before using the X25519 or X448 function defined in Section 5 of [RFC7748]. For further details, see Section 2.5.2. Elliptic Curve Cryptography (ECC) asymmetric keys in Montgomery or Weierstrass form are used directly in the key agreement algorithm without coordinate mapping.

As long as any two group members preserve the same asymmetric keys, their Diffie-Hellman shared secret does not change across updates of the group keying material. The lifetime of those keys is the same as the lifetime of the authentication credentials Sender Auth Cred and Recipient Auth Cred.

After establishing a partially or completely new Security Context (see Sections 2.6 and 12.2), the old pairwise keys **MUST** be deleted. Since new Sender/Recipient Keys are derived from the new group keying material (see Section 2.2), every group member **MUST** use the new Sender/Recipient Keys when deriving new pairwise keys.

2.5.2. ECDH with Montgomery Coordinates

2.5.2.1. Curve25519

The y-coordinate of the other endpoint's Ed25519 public key is decoded as specified in Section 5.1.3 of [RFC8032]. The Curve25519 u-coordinate is recovered as u = (1 + y) / (1 - y) (mod p) following the map in Section 4.1 of [RFC7748]. Note that the mapping is not defined for y = 1, and that y = -1 maps to u = 0, which corresponds to the neutral group element and thus will result in a degenerate shared secret. Therefore, implementations **MUST** abort if the y-coordinate of the other endpoint's Ed25519 public key is 1 or -1 (mod p).

The private signing key byte strings (i.e., the lower 32 bytes used for generating the public key; see Step 1 of Section 5.1.5 of [RFC8032]) are decoded the same way for signing in Ed25519 and scalar multiplication in X25519. Hence, in order to compute the shared secret, the endpoint applies the X25519 function to the Ed25519 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.5.2.2. Curve448

The y-coordinate of the other endpoint's Ed448 public key is decoded as specified in Section 5.2.3 of [RFC8032]. The Curve448 u-coordinate is recovered as $u = y^2 * (d * y^2 - 1) / (y^2 - 1) \pmod p$ following the map from "edwards448" in Section 4.2 of [RFC7748], and also using the relation $x^2 = (y^2 - 1)/(d * y^2 - 1)$ from the curve equation. Note that the mapping is not defined for y = 1 or -1. Therefore, implementations **MUST** abort if the y-coordinate of the peer endpoint's Ed448 public key is 1 or -1 (mod p).

The private signing key byte strings (i.e., the lower 57 bytes used for generating the public key, see Step 1 of Section 5.2.5 of [RFC8032]) are decoded the same way for signing in Ed448 and scalar multiplication in X448. Hence, in order to compute the shared secret, the endpoint applies the X448 function to the Ed448 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.5.3. Usage of Sequence Numbers

When using any of its Pairwise Sender Keys, a sender endpoint including the 'Partial IV' parameter in the protected message **MUST** use the current fresh value of the Sender Sequence Number from its Sender Context (see Section 2.2). That is, the same Sender Sequence Number space is used for all outgoing messages protected with Group OSCORE, thus limiting both storage and complexity.

When combining communications with the group mode and the pairwise mode, this may result in the Partial IV values moving forward more often than when using OSCORE [RFC8613]. This can happen when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, by sending requests over unicast. In turn, this contributes to a sooner exhaustion of the Sender Sequence Number space of the client, which would then require to take actions for deriving a new Sender Context before resuming communications in the group (see Section 2.6.2).

2.5.4. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes the Pairwise Key Agreement Algorithm and the pairwise keys as described at the beginning of Section 2.

The pairwise keys, as well as the shared secrets used in their derivation (see Section 2.5.1), may be stored in memory or recomputed every time they are needed. The shared secret only changes when a public/private key pair used for its derivation changes, which results in the pairwise keys also changing. Additionally, the pairwise keys change if the Sender ID changes or if a new Security Context is established for the group (see Section 2.6.3). In order to optimize protocol performance, an endpoint may store the derived pairwise keys for easy retrieval.

In the pairwise mode, the Sender Context includes the Pairwise Sender Keys to use with the other endpoints (see Figure 1). In order to identify the right key to use, the Pairwise Sender Key for endpoint X may be associated with the Recipient ID of endpoint X, as defined in the Recipient Context (i.e., the Sender ID from the point of view of endpoint X). In this way, the Recipient ID can

be used to lookup for the right Pairwise Sender Key. This association may be implemented in different ways, e.g., by storing the pair (Recipient ID, Pairwise Sender Key) or linking a Pairwise Sender Key to a Recipient Context.

2.6. Update of Security Context

It is **RECOMMENDED** that the long-term part of the Security Context is stored in non-volatile memory, or that it can otherwise be reliably accessed throughout the operation of the group, e.g., after device reboots. However, data in the long-term part of the Security Context may also need to be updated, for example, due to scheduled key renewal, new or re-joining members in the group, or the fact that the endpoint changes Sender ID (see Section 2.6.3).

The data in the varying part of the Security Context are updated by the endpoint when executing the security protocol but may be lost (see Section 2.6.1) or become outdated by exhaustion of Sender Sequence Numbers (see Section 2.6.2).

2.6.1. Loss of the Varying Part of the Security Context

An endpoint may lose the varying part of its Security Context due to accidental events, e.g., if a reboot occurred in an unprepared way (see Section 2.6.1.1) or due to a deliberately deleted Recipient Context (see Section 2.6.1.2).

If it is not feasible or practically possible to store and maintain up-to-date the varying part in non-volatile memory (e.g., due to limited number of write operations), the endpoint **MUST** be able to detect a loss of the varying part of the Security Context in order to prevent the reuse of a nonce with the same key and to handle incoming replayed messages.

2.6.1.1. Accidental Loss of Sender Context and/or Recipient Contexts

In the case that a loss of the Sender Context and/or of the Recipient Contexts is detected (e.g., if a reboot occurred in an unprepared way), the endpoint **MUST NOT** protect further messages using this Security Context to avoid reusing a nonce with the same key.

Before resuming its operations in the group, the endpoint **MUST** retrieve new Security Context parameters from the Group Manager (see Section 2.6.3) and use them to derive a new Sender Context and new Recipient Contexts (see Section 2.2). Since the new Sender Context includes newly derived encryption keys, an endpoint will not reuse the same pair (key, nonce), even when it is a server using the Partial IV of (old re-injected) requests to build the nonce for protecting the responses.

From then on, the endpoint **MUST** use the latest installed Sender Context to protect outgoing messages. Newly derived Recipient Contexts will have a Replay Window that is initialized as valid.

If an endpoint is not configured as a silent server and is not able to establish an updated Sender Context, e.g., because of lack of connectivity with the Group Manager, then the endpoint **MUST NOT** protect further messages using the current Security Context and **MUST NOT** accept incoming messages from other group members, as it is currently unable to detect possible replays.

If an endpoint is configured as a silent server and is not able to establish an updated Security Context, e.g., because of lack of connectivity with the Group Manager, then the endpoint **MUST NOT** accept incoming messages from other group members, as it is currently unable to detect possible replays.

2.6.1.2. Deliberately Deleted Recipient Contexts

The Security Context may contain a large and variable number of Recipient Contexts. While Group OSCORE does not establish a maximum number of Recipient Contexts, there are circumstances by which implementations might choose to discard Recipient Contexts or have to do so in accordance with enforced application policies. Such circumstances include the need to reclaim memory or other resources on the node hosting the endpoint, for example, because the predefined maximum number of Recipient Contexts has been reached in the Security Context (see Section 2.2). Implementations can provide means for the application to gain knowledge about the deliberate deletion of Recipient Contexts, e.g., through notifications sent to the application and/or logs made available to the application.

When a Recipient Context is deleted, this not only results in losing information about previously received messages from the corresponding other endpoint, but it also results in the inability to be aware of the Security Contexts from which information has been lost.

Therefore, if the Recipient Context is derived again from the same Security Context, there is a risk that a replayed message is not detected. If any Recipient Context associated with any peer has ever been deleted from the current Security Context, then the Replay Window of any new Recipient Context in this Security Context **MUST** be initialized as invalid. An exception applies when the deleted Recipient Context was created upon receiving a message and that message was not verified successfully (see Sections 7.2, 7.4, 8.4, and 8.6). Messages associated with a Recipient Context that has an invalid Replay Window **MUST NOT** be delivered to the application.

If the endpoint receives a message to be processed with any such new Recipient Context whose Replay Window is invalid, then the endpoint **MUST** take one of the following courses of action:

- The endpoint discards the message.
- The endpoint follows the procedure based on the CoAP Echo Option [RFC9175] and specified in Section 9 in order to establish a valid Replay Window. In particular, the endpoint **MUST** use its Partial IV when generating the nonce and **MUST** include the Partial IV in the response message conveying the Echo Option. If the endpoint supports the CoAP Echo Option, then it is **RECOMMENDED** to take this course of action.
- The endpoint retrieves or waits for new Security Context parameters from the Group Manager and derives new Sender and Recipient Contexts as defined in Section 2.6.1.1. In this case, the Replay Windows of all Recipient Contexts become valid if they are not already. In particular, any invalid Replay Window is re-initialized as valid and with 0 as its current lower limit.

2.6.2. Exhaustion of Sender Sequence Number Space

Since an endpoint increments its Sender Sequence Number for each new outgoing message including a Partial IV, an endpoint can eventually exhaust the Sender Sequence Number space.

Implementations **MUST** be able to detect an exhaustion of Sender Sequence Number space after the endpoint has consumed the largest usable value. This may be influenced by additional limitations besides the mere 40-bit size limit of the Partial IV.

Upon exhausting the Sender Sequence Number space, the endpoint **MUST NOT** use this Security Context to protect further messages including a Partial IV.

When approaching the exhaustion of the Sender Sequence Number space, the endpoint **SHOULD** inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.6.3), and use them to derive a new Sender Context (see Section 2.2). It is **RECOMMENDED** that the endpoint takes this course of action with some margin, i.e., well before exhausting the Sender Sequence Number space, in order to avoid a period of inability to protect messages including a Partial IV.

From then on, the endpoint **MUST** use its latest installed Sender Context to protect outgoing messages.

2.6.3. Retrieving New Security Context Parameters

The Group Manager can assist an endpoint with an incomplete Sender Context to retrieve missing data of the Security Context and thereby become fully operational in the group again. The two main options for the Group Manager are i) assignment of a new Sender ID to the endpoint (see Section 2.6.3.1), and ii) establishment of a new Security Context for the group (see Section 2.6.3.2). The update of the Replay Window in each of the Recipient Contexts is discussed in Section 2.6.1.

As group membership changes, or as group members get new Sender IDs (see Section 2.6.3.1), so do the relevant Recipient IDs that the other endpoints need to keep track of. As a consequence, group members may end up retaining stale Recipient Contexts that are no longer useful to verify incoming secure messages.

The Recipient ID ('kid') **SHOULD NOT** be considered as a persistent and reliable identifier of a group member. Such an indication can be achieved only by using that member's public key, when verifying countersignatures of received messages (in group mode), or when verifying messages integrity-protected with pairwise keying material derived from authentication credentials and associated asymmetric keys (in pairwise mode).

Furthermore, applications **MAY** define policies to delete (long-)unused Recipient Contexts and reduce the impact on storage space, and to check with the Group Manager that an authentication credential with the public key included therein is currently the one associated with a 'kid' value after a number of consecutive failed verifications.

2.6.3.1. New Sender ID for the Endpoint

The Group Manager may assign a new Sender ID to an endpoint, while leaving the Gid, Master Secret, and Master Salt unchanged in the group. In this case, the Group Manager assigns a Sender ID that has not been used in the group since the latest time when the current Gid value was assigned to the group (see Section 12.2).

Having retrieved the new Sender ID, and potentially other missing data for the long-term part of the Security Context, the endpoint can derive a new Sender Context (see Section 2.2). When doing so, the endpoint resets the Sender Sequence Number in its Sender Context to 0 and derives a new Sender Key. This is, in turn, used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint **MUST** use its latest installed Sender Context to protect outgoing messages.

The assignment of a new Sender ID may be the result of different processes. The endpoint may request a new Sender ID, e.g., because of the impending exhaustion of the Sender Sequence Number space (see Section 2.6.2). An endpoint may request to re-join the group, e.g., because of losing the varying part of its Security Context (see Section 2.6.1), and is provided with a new Sender ID together with the latest data for the long-term part of the Security Context.

For the other group members, the Recipient Context corresponding to the old Sender ID becomes stale (see Section 12.2).

2.6.3.2. New Security Context for the Group

The Group Manager may establish a new Security Context for the group (see Section 12.2). The Group Manager does not necessarily establish a new Security Context for the group if one member has an outdated Security Context (see Section 2.6.3.1) unless that was already planned or required for other reasons.

All the group members need to acquire new Security Context parameters from the Group Manager. Once having acquired new Security Context parameters, each group member performs the following actions:

- From then on, it **MUST NOT** use the current Security Context to start processing new messages for the considered group.
- It completes any ongoing message processing for the considered group.
- It derives and installs a new Security Context. In particular:
  - It re-derives the keying material stored in its Sender Context and Recipient Contexts (see Section 2.2). The Master Salt used for the re-derivations is the updated Master Salt parameter if provided by the Group Manager, or the empty byte string otherwise.
  - It resets its Sender Sequence Number in its Sender Context to 0.
  - It re-initializes the Replay Window of each Recipient Context as valid and with 0 as its current lower limit.
  - For each long exchange where it is a client and that it wants to keep active, it sets the Response Number of each associated server as not initialized (see Section 5.1).

From then on, it can resume processing new messages for the considered group. In particular:

- It **MUST** use its latest installed Sender Context to protect outgoing messages.
- It **SHOULD** use only its latest installed Recipient Contexts to process incoming messages, unless application policies admit to temporarily retain and use the old and recent Security Context (see Section 13.5.1).

The distribution of a new Gid and Master Secret may result in temporarily misaligned Security Contexts among group members. In particular, this may result in a group member not being able to process messages received right after a new Gid and Master Secret have been distributed. A discussion on practical consequences and possible ways to address them, as well as on how to handle the old Security Context, is provided in Section 13.5.

3. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [RFC9052] to wrap and protect data in the original message. Like OSCORE, Group OSCORE uses the untagged COSE_Encrypt0 structure with an AEAD algorithm. Unless otherwise specified, the following modifications to what is defined for OSCORE apply for both the group mode and the pairwise mode of Group OSCORE.

3.1. Countersignature

When protecting a message in group mode, the 'unprotected' field **MUST** additionally include the following parameter:

Countersignature0 version 2: Its value is set to the countersignature of the COSE object.

The countersignature is computed by the sender as described in Sections 3.2 and 3.3 of [RFC9338] by using its private key and according to the Signature Algorithm in the Security Context.

In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 3.4 of this document, and the ciphertext of the COSE object as payload.

3.1.1. Clarifications on Using a Countersignature

The literature commonly refers to a countersignature as a signature computed by an entity A over a document already protected by a different entity B.

However, the COSE_Countersignature0 structure belongs to the set of abbreviated countersignatures defined in Sections 3.2 and 3.3 of [RFC9338], which were designed primarily to deal with the problem of encrypted group messaging, but where it is required to know who originated the message.

Since the parameters for computing or verifying the abbreviated countersignature generated by A are provided by the same context used to describe the security processing performed by B and to be countersigned, these structures are applicable also when the two entities A and B are actually the same one, like the sender of a Group OSCORE message protected in group mode.

3.2. The 'kid' and 'kid context' Parameters

The value of the 'kid' parameter in the 'unprotected' field of response messages **MUST** be set to the Sender ID of the endpoint transmitting the message if the request was protected in group mode. That is, unlike in [RFC8613], the 'kid' parameter is always present in responses to a request that was protected in group mode.

The value of the 'kid context' parameter in the 'unprotected' field of request messages **MUST** be set to the ID Context, i.e., the Gid of the group. That is, unlike in [RFC8613], the 'kid context' parameter is always present in requests.

3.3. Nonce Computation

The nonce is constructed like in OSCORE, with the difference that Step 4 in Section 5.2 of [RFC8613] is replaced with:

4. and then XOR with X bytes from the Common IV's start, where X is the length in bytes of the nonce.

For example, if X = 7 and the Common IV is 0x00112233445566778899aabbcc (13 bytes), then the bytes to XOR are 0x00112233445566 (7 bytes).

The constructed nonce is used both by the AEAD Algorithm (see Section 2.1.1) and by the Group Encryption Algorithm (see Section 2.1.7), independent of whether they are AEAD or plain encryption algorithms. Algorithms that do not use a nonce are not supported, as per Section 2.1.7.

3.4. Additional Authenticated Data

The external_aad of the AAD is different compared to OSCORE [RFC8613] and is defined in this section.

The same external_aad structure is used in group mode and pairwise mode for encryption/decryption (see Section 5.3 of [RFC9052]), as well as in group mode for computing and verifying the countersignature (see Sections 3.2 and 3.3 of [RFC9338]).

In particular, the external_aad includes also the Signature Algorithm, the Group Encryption Algorithm, the Pairwise Key Agreement Algorithm, the value of the 'kid context' in the COSE object of the request, the OSCORE Option of the protected message, the sender's authentication credential, and the Group Manager's authentication credential.

The external_aad **SHALL** be a CBOR array wrapped in a bstr object as defined below, following the notation of [RFC8610]:

```
external_aad = bstr .cbor aad_array

aad_array = [
   oscore_version : uint,
   algorithms : [alg_aead : int / tstr / null,
                 alg_group_enc : int / tstr / null,
                 alg_signature : int / tstr / null,
                 alg_pairwise_key_agreement : int / tstr / null],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr,
   OSCORE_option : bstr,
   sender_cred : bstr,
   gm_cred : bstr
]
```

Figure 2: external_aad

Compared with Section 5.4 of [RFC8613], the aad_array has the following differences:

- The 'algorithms' array is extended as follows.

  The parameter 'alg_aead' **MUST** be set to the CBOR simple value `null` (0xf6) if the parameter AEAD Algorithm is not set in the Common Context of the Security Context used (see Section 2.1.1). Otherwise, regardless of whether the endpoint supports the pairwise mode or not, this parameter **MUST** specify AEAD Algorithm from the Common Context (see Section 2.1.1) as per Section 5.4 of [RFC8613].

  Furthermore, the 'algorithms' array additionally includes:

  - 'alg_group_enc', which specifies Group Encryption Algorithm from the Common Context of the Security Context used (see Section 2.1.7). This parameter **MUST** be set to the CBOR simple value `null` (0xf6) if the parameter Group Encryption Algorithm in the Common Context is not set. Otherwise, regardless of whether the endpoint supports the group mode or not, this parameter **MUST** specify Group Encryption Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" registry for this algorithm.
  - 'alg_signature', which specifies Signature Algorithm from the Common Context of the Security Context used (see Section 2.1.8). This parameter **MUST** be set to the CBOR simple value `null` (0xf6) if the parameter Signature Algorithm in the Common Context is not set. Otherwise, regardless of whether the endpoint supports the group mode or not, this parameter **MUST** specify Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" registry for this algorithm.
  - 'alg_pairwise_key_agreement', which specifies Pairwise Key Agreement Algorithm from the Common Context of the Security Context used (see Section 2.1.10). This parameter **MUST** be set to the CBOR simple value `null` (0xf6) if the parameter Pairwise Key Agreement Algorithm in the Common Context is not set. Otherwise, regardless of whether

the endpoint supports the pairwise mode or not, this parameter **MUST** specify Pairwise Key Agreement Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" registry for this algorithm.

- The new element 'request_kid_context' contains the value of the 'kid context' in the COSE object of the request (see Section 3.2).

  This enables endpoints to safely keep a long exchange active beyond a possible change of Gid (i.e., of ID Context), following a group rekeying (see Section 12.2). In fact, it ensures that every response within a long exchange cryptographically matches with only one request (i.e., the request associated with that long exchange) rather than with multiple requests that were protected with different keying material but share the same 'request_kid' and 'request_piv' values.

- The new element 'OSCORE_option', containing the value of the OSCORE Option present in the protected message, encoded as a binary string. This prevents the attack described in Section 13.7 when using the group mode, as further explained in Section 13.7.2.

  Note for implementation: This construction requires the OSCORE Option of the message to be generated and finalized before computing the ciphertext of the COSE_Encrypt0 object (when using the group mode or the pairwise mode) and before calculating the countersignature (when using the group mode). Also, the aad_array needs to be large enough to contain the largest possible OSCORE Option.

- The new element 'sender_cred', containing the sender's authentication credential. This parameter **MUST** be set to a CBOR byte string, which encodes the sender's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.4).
- The new element 'gm_cred', containing the Group Manager's authentication credential. This parameter **MUST** be set to a CBOR byte string, which encodes the Group Manager's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.4). This prevents the attack described in Section 13.8.

4. OSCORE Header Compression

Group OSCORE relies on a header compression mechanism similar to the one used by OSCORE and specified in Section 4.1. Examples are provided in Section 4.3.

4.1. Encoding of the OSCORE Option Value and Group OSCORE Payload

The OSCORE header compression defined in Section 6 of [RFC8613] is used for compactly encoding the COSE_Encrypt0 object specified in Section 3 of this document with the following differences:

- When the Group OSCORE message is protected in group mode, the message payload **SHALL** encode the ciphertext of the COSE object, concatenated with the encrypted countersignature of the COSE object. That is:
  - The plain, original countersignature of the COSE object, namely SIGNATURE, is specified in the "Countersignature0 version 2" parameter within the 'unprotected' field of the COSE object (see Section 3.1).
  - The encrypted countersignature, namely ENC_SIGNATURE, is computed as

    ENC_SIGNATURE = SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 4.2.
- When the Group OSCORE message is protected in pairwise mode, the message payload **SHALL** encode the ciphertext of the COSE object.
- This document defines the usage of the sixth least significant bit, called "Group Flag", in the first byte of the OSCORE Option value containing the OSCORE flag bits. This flag bit is specified in Section 14.1.
- The Group Flag **MUST** be set to 1 if the Group OSCORE message is protected using the group mode (see Section 7).
- The Group Flag **MUST** be set to 0 if the Group OSCORE message is protected using the pairwise mode (see Section 8). The Group Flag **MUST** also be set to 0 for ordinary OSCORE messages processed according to [RFC8613].

4.2. Keystream Derivation for Countersignature Encryption/Decryption

This section defines how an endpoint derives the keystream KEYSTREAM, which is used to encrypt/decrypt the countersignature of an outgoing/incoming message M protected in group mode.

The keystream **SHALL** be derived as follows by using the HKDF Algorithm from the Common Context (see Section 2.1.2) analogously to Section 3.2.1 of [RFC8613], i.e., by composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

KEYSTREAM = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows:

- salt takes as its value the Partial IV (PIV) used to protect M. Note that, if M is a response, salt takes as its value either the fresh Partial IV generated by the server and included in the response or the same Partial IV of the request generated by the client and not included in the response.

- IKM is the Signature Encryption Key from the Common Context (see Section 2.1.9).
- info is the serialization of a CBOR array with the structure defined below, following the notation of [RFC8610]:

```
info = [
  id : bstr,
  id_context : bstr,
  type : bool,
  L : uint
]
```

where:

- id is the Sender ID of the endpoint that generated PIV.
- id_context is the ID Context (Gid) used when protecting M.

  Note that, in the case of group rekeying, a server might use a different Gid when protecting a response compared to the Gid that it used to verify (that the client used to protect) the request; see Section 7.3.
- type is the CBOR simple value `true` (0xf5) if M is a request or the CBOR simple value `false` (0xf4) otherwise.
- L is the size of the countersignature, as per Signature Algorithm from the Common Context (see Section 2.1.8) in bytes.

4.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE header compression examples of Group OSCORE used in group mode (see Section 4.3.1) or in pairwise mode (see Section 4.3.2).

The examples assume that the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the examples do not include the full CoAP unprotected message or the full Security Context, but only the input necessary to the compression mechanism, i.e., the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 4.1 and divided into two parts, since the object is transported in two CoAP fields: OSCORE Option and payload.

The examples assume that the plaintext (see Section 5.3 of [RFC8613]) is 6 bytes long, and that the AEAD tag is 8 bytes long, hence resulting in a ciphertext that is 14 bytes long. When using the group mode, the COSE_Countersignature0 byte string as described in Section 3 is assumed to be 64 bytes long.

4.3.1. Examples in Group Mode

Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5, and kid context = 0x44616c.

- Before compression (96 bytes):

```
[
  / protected / h'',
  / unprotected / {
    / kid /                          4 : h'25',
    / Partial IV /                   6 : h'05',
    / kid context /                 10 : h'44616c',
    / Countersignature0 version 2 /  12 : h'66e6d9b0
    db009f3e105a673f8855611726caed57f530f8cae9d0b168
    513ab949fedc3e80a96ebe94ba08d3f8d3bf83487458e2ab
    4c2f936ff78b50e33c885e35'
  },
  / ciphertext / h'aea0155667924dff8a24e4cb35b9'
]
```

- After compression (85 bytes):

```
Flag byte: 0b00111001 = 0x39 (1 byte)

Option Value: 0x39 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 de9e ... f1
(14 bytes + size of the encrypted countersignature)
```

Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52, and no Partial IV.

- Before compression (88 bytes):

```
[
  / protected / h'',
  / unprotected / {
    / kid /                          4 : h'52',
    / Countersignature0 version 2 /  12 : h'f5b659b8
    24487eb349c5c5c8a3fe401784cade2892725438e8be0fab
    daa2867ee6d29f68edb0818e50ebf98c28b923d0205f5162
    e73662e27c1a3ec562a49b80'
  },
  / ciphertext / h'60b035059d9ef5667c5a0710823b'
]
```

- After compression (80 bytes):

```
Flag byte: 0b00101000 = 0x28 (1 byte)

Option Value: 0x28 52 (2 bytes)

Payload: 0x60b035059d9ef5667c5a0710823b ca1e ... b3
(14 bytes + size of the encrypted countersignature)
```

4.3.2. Examples in Pairwise Mode

Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5, and kid context = 0x44616c.

- Before compression (29 bytes):

```
[
  / protected / h'',
  / unprotected / {
    / kid /           4 : h'25',
    / Partial IV /    6 : h'05',
    / kid context /  10 : h'44616c'
  },
  / ciphertext / h'aea0155667924dff8a24e4cb35b9'
]
```

- After compression (21 bytes):

```
Flag byte: 0b00011001 = 0x19 (1 byte)

Option Value: 0x19 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)
```

Response with ciphertext = 0x60b035059d9ef5667c5a0710823b and no Partial IV.

- Before compression (18 bytes):

```
[
  / protected / h'',
  / unprotected / {},
  / ciphertext / h'60b035059d9ef5667c5a0710823b'
]
```

- After compression (14 bytes):

```
Flag byte: 0b00000000 = 0x00 (1 byte)

Option Value: 0x (0 bytes)

Payload: 0x60b035059d9ef5667c5a0710823b (14 bytes)
```

5. Message Binding, Sequence Numbers, Freshness, and Replay Protection

Like OSCORE, Group OSCORE provides message binding of responses to requests, as well as uniqueness of (key, nonce) pairs (see Sections 7.1 and 7.2 of [RFC8613], respectively).

5.1. Supporting Multiple Responses in Long Exchanges

For each of its ongoing long exchanges, a client maintains one Response Number for each different server. Then, separately for each server, the client uses the associated Response Number to perform ordering and replay protection of responses received from that server within that long exchange (see Section 5.3.1).

That is, the Response Number has the same purpose that the Notification Number has in OSCORE (see Section 4.1.3.5.2 of [RFC8613]), but a client uses it for handling any response from the associated server within a long exchange.

Group OSCORE allows a long exchange to remain active, even if the group is rekeyed (thus changing the ID Context) or the client obtains a new Sender ID.

As defined in Section 7, this is achieved by the client and server(s) storing the 'kid' and 'kid context' used in the original request throughout the whole duration of the long exchange.

Upon leaving the group or before re-joining the group, a group member **MUST** terminate all the ongoing long exchanges that it has started in the group as a client. This frees up the CoAP Token associated with the corresponding request.

5.2. Freshness

If the application requires freshness, e.g., according to time- or event-based policies (see Section 2.5.1 of [RFC9175]), a server can use the approach in Section 9 as a variant of the Challenge-Response procedure based on the Echo Option [RFC9175] before delivering request messages from a client to the application.

Like in OSCORE [RFC8613], assuming an honest server, the message binding guarantees that a response is not older than the request it replies to. Therefore, building on Section 7.3 of [RFC8613], the following properties apply for Group OSCORE:

- The freshness of a response can be assessed if it is received soon after the request.

  For responses within a long exchange, this assessment gets weaker with time. If such responses are Observe notifications [RFC7641], it is **RECOMMENDED** that the client regularly re-registers the observation.

  If the request was neither a group request nor an Observe request, there is at most a single valid response and only from one individually targeted server in the group. Thus, freshness can be assessed depending on when the request was sent.

- It is not guaranteed that a misbehaving server did not create the response before receiving the request, i.e., Group OSCORE does not verify server aliveness.
- For requests and responses, the received Partial IV allows a recipient to determine the relative order of requests or responses.

5.3. Replay Protection

Like in OSCORE [RFC8613], the replay protection relies on the Partial IV of incoming messages. A server updates the Replay Window of its Recipient Contexts based on the Partial IV values in received request messages, which correspond to the Sender Sequence Numbers of the clients. Note that there can be large jumps in these Sender Sequence Number values, e.g., when a client exchanges unicast messages with other servers. The operation of validating the Partial IV and performing replay protection **MUST** be atomic. Sections 2.6.1 and 2.6.3.2 describe the update of Replay Windows after the loss of data from the Security Context and the retrieving of new Security Context parameters.

The protection from replay of requests is performed as per Section 7.4 of [RFC8613] separately for each client and by leveraging the Replay Window in the corresponding Recipient Context. The protection from replay of responses in a long exchange is performed as defined in Section 5.3.1 of this document.

5.3.1. Replay Protection of Responses

A client uses the method defined in this section in order to check whether a received response is a replay.

This especially applies to responses received within a long exchange during which multiple such responses can be received from the same server to the corresponding request. These include Observe notifications [RFC7641] and non-notification responses as a reply to a group request, which the client can receive until the CoAP Token value associated with the group request is freed up (see Section 3.1.6 of [RFC10020]).

When sending a response (both successful and error), a server **MUST** include its Sender Sequence Number as Partial IV in the response except when sending the first response to the corresponding request, in which case, the Partial IV in the response **MAY** be omitted.

In order to protect against replay, the client **SHALL** maintain for each ongoing long exchange one Response Number for each different server. The Response Number is a non-negative integer containing the largest Partial IV of the responses received from that server during the long exchange while using the same Security Context.

Then, separately for each server, the client uses the associated Response Number to perform ordering and replay protection of the responses from that server during the long exchange by comparing their Partial IVs with one another and against the Response Number.

For every long exchange, the Response Number associated with a server is initialized to the Partial IV of the response from that server such that, within the long exchange, it is the first response from that server to include a Partial IV and to be successfully verified with the used

Security Context. Note that, when a new Security Context is established in the group, the client sets the Response Number of each associated server as not initialized (see Section 2.6.3.2), hence later responses within the same long exchange and protected with the new Security Context will result in a new initialization of Response Numbers. Furthermore, for every long exchange, a client **MUST** only accept at most one response without Partial IV from each server and treat it as the oldest response from that server within that long exchange.

During a long exchange, a client receiving a response containing a Partial IV **SHALL** compare the Partial IV with the Response Number associated with the replying server within that long exchange. The client **MUST** stop processing a response from a server if that response has a Partial IV that has been previously received from that server during that long exchange, while using the same Security Context.

Applications **MAY** decide that a client only processes responses within a long exchange if those have a greater Partial IV than the Response Number associated with the replying server within that long exchange. This limits the storage overhead on the client to maintaining one Response Number per replying server within the long exchange. Conversely, more permissive applications can allow clients to also process responses that have a smaller Partial IV than the Response Number associated with the replying server. For a client, the ability to detect previously received Partial IVs while admitting the processing of such responses comes at the cost of additional storage overhead, for which a reasonable bound has to be defined by the application. A possible way to achieve that relies on using a sliding Replay Window uniquely paired with the replying server within the long exchange, similarly to that used by a server for detecting replayed requests.

If the verification of the response succeeds, and the received Partial IV (when included) was greater than the Response Number associated with the replying server, then the client **SHALL** overwrite that Response Number with the received Partial IV.

As long as a server uses the same Security Context to protect its responses to the same request, the client **MUST** consider the response with the highest Partial IV as the freshest response from that server among those protected with that Security Context, regardless of the order of arrival. Within a long exchange, implementations need to make sure that a response without Partial IV is considered the oldest response from the replying server within that long exchange.

The method defined in this section is not relevant for responses to requests that are neither group requests nor Observe requests. In fact, for each of such requests, there is at most one response and only from one individually targeted server in the group.

6. Message Reception

Upon receiving a protected message, a recipient endpoint retrieves a Security Context as in [RFC8613]. An endpoint **MUST** be able to distinguish between a Security Context to process OSCORE messages as in [RFC8613] and a Group OSCORE Security Context to process Group OSCORE messages as defined in this document.

The way to accomplish this distinction is implementation specific. For example, an endpoint can take into account the different structure of the Security Context defined in Section 2, e.g., based on the presence of Signature Algorithm and Pairwise Key Agreement Algorithm in the Common Context. Alternatively, at the cost of increasing storage, implementations can use an additional parameter in the Security Context to explicitly mark that it is intended for processing Group OSCORE messages.

If either of the following conditions applies, a recipient endpoint **MUST** discard the incoming protected message:

- The Group Flag is set to 0 and the retrieved Security Context is associated with an OSCORE group, but the endpoint does not support the pairwise mode or any of the following parameters is not set in the Security Context: the AEAD Algorithm and the Pairwise Key Agreement Algorithm.
- The Group Flag is set to 1 and the retrieved Security Context is associated with an OSCORE group, but the endpoint does not support the group mode or any of the following parameters is not set in the Security Context: the Group Encryption Algorithm and the Signature Algorithm.
- The Group Flag is set to 1, but there is no Security Context associated with an OSCORE group.

  Future specifications may define how to process incoming messages protected with Security Contexts as in [RFC8613] when the Group Flag bit is set to 1.

Otherwise, if a Security Context associated with an OSCORE group is retrieved, the recipient endpoint processes the message with Group OSCORE using the group mode (see Section 7) if the Group Flag is set to 1 or the pairwise mode (see Section 8) if the Group Flag is set to 0.

Note that if the Group Flag is set to 0 and the recipient endpoint retrieves a Security Context that is valid to process the message but is not associated with an OSCORE group, then the message is processed according to [RFC8613].

7. Message Processing in Group Mode

When using the group mode, messages are protected and processed as specified in [RFC8613] with the modifications described in this section. The security objectives of the group mode are discussed in Appendix A.2.

The possible use of the group mode is indicated by the Group Manager as part of the group data provided to new group members when joining the group, according to which the parameters Group Encryption Algorithm and Signature Algorithm in the Security Context are set (see Section 2).

During all the steps of the message processing, an endpoint **MUST** use the same Security Context for the considered group. That is, an endpoint **MUST NOT** install a new Security Context for that group (see Section 2.6.3.2) until the message processing is completed.

The group mode **SHOULD** be used to protect group requests intended for multiple recipients or for the whole group. This applies to both requests directly addressed to multiple recipients, e.g., sent by the client over multicast, as well as requests sent by the client over unicast to a proxy that forwards them to the intended recipients over multicast [RFC10020]. Exceptions where the requirement above is not fulfilled and the pairwise mode is used to protect group requests include: the efficient discovery of a server's address in the group (see Section 8.1) or the enabling of simple constructions where a variation of the pairwise mode protects requests possibly intended to multiple servers in such a way that the corresponding responses are effectively cacheable by intermediaries (e.g., see [CACHEABLE-OSCORE]).

As per [RFC7252] and [RFC10020], group requests sent over multicast are always Non-confirmable and thus are not retransmitted by the CoAP messaging layer. Instead, applications should store such outgoing messages for a predefined, sufficient amount of time in order to correctly perform potential retransmissions at the application layer. If performed, these retransmissions are repetitions of previous protected messages, which the sender endpoint does not protect again with Group OSCORE.

According to Section 5.2.3 of [RFC7252]:

> If the request message is Non-confirmable, then the response **SHOULD** be returned in a Non-confirmable message as well. However, an endpoint **MUST** be prepared to receive ... a Confirmable response in reply to a Non-confirmable request.

Confirmable group requests are acknowledged when sent over non-multicast transports as specified in [RFC7252].

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [RFC8613]. In addition, a recipient **MUST** stop processing and reject any message that is malformed and that does not follow the format specified in Section 3 of this document or that is not cryptographically validated in a successful way as per the processing defined in Sections 7.2 and 7.4.

In either case, it is **RECOMMENDED** that a server does not send back any error message in reply to a received request if either of the following conditions applies:

- The server is not able to identify whether the received request is a group request, i.e., as sent to all servers in the group.
- The server identifies the received request as a group request.

This prevents servers from replying with multiple error messages to a client sending a group request, avoiding the risk of flooding and possibly congesting the network.

7.1. Protecting the Request

When using the group mode to protect a request, a client proceeds as described in Section 8.1 of [RFC8613] with the following modifications.

- In Step 2, the AAD is modified as described in Section 3.4 of this document.
- In Step 4, the encryption of the COSE object is modified as described in Section 3 of this document. The encoding of the compressed COSE object is modified as described in Section 4 of this document. In particular, the Group Flag **MUST** be set to 1. The Group Encryption Algorithm from the Common Context **MUST** be used.
- In Step 5, the countersignature is computed and the format of the OSCORE message is modified as described in Sections 3 and 4 of this document. In particular, the payload of the Group OSCORE message includes also the encrypted countersignature.

In addition, the following applies when sending a request that establishes a long exchange:

- If the client intends to keep the long exchange active beyond a possible change of Sender ID, the client **MUST** store the value of the 'kid' parameter from the request and retain it until the long exchange is terminated. Even in the case that the client is individually rekeyed and receives a new Sender ID from the Group Manager (see Section 2.6.3.1), the client **MUST NOT** update the stored 'kid' parameter value associated with the long exchange and the corresponding request.
- If the client intends to keep the long exchange active beyond a possible change of ID Context following a group rekeying (see Section 12.2), then the following applies:
  - The client **MUST** store the value of the 'kid context' parameter from the request and retain it until the long exchange is terminated. Upon establishing a new Security Context with a new Gid as ID Context (see Section 2.6.3.2), the client **MUST NOT** update the stored 'kid context' parameter value associated with the long exchange and the corresponding request.
  - The client **MUST** store an invariant identifier of the group, which is immutable even if the Security Context of the group is re-established. For example, this invariant identifier can be the "group name" in [ACE-KEY-GROUP-OSCORE], where it is used for joining the group and retrieving the current group keying material from the Group Manager.

    After a group rekeying, the client might have missed both the rekeying messages and the servers' first responses that are protected with the new Security Context and include the new ID Context (Gid) in the 'kid context' parameter (see Section 7.3). In such a case, while still not knowing the new ID Context (Gid) used in the group, the client is able to retrieve the current group keying material from the Group Manager using the invariant identifier to unambiguously refer to the group.

7.2. Verifying the Request

Upon receiving a protected request with the Group Flag set to 1, following the procedure in Section 6, a server proceeds as described in Section 8.2 of [RFC8613] with the following modifications.

- In Step 2, the decoding of the compressed COSE object follows Section 4 of this document. In particular:
  - If the server discards the request due to not retrieving a Security Context associated with the OSCORE group, the server **MAY** respond with a 4.01 (Unauthorized) error message. When doing so, the server **MAY** set an Outer Max-Age Option with value zero and **MAY** include a descriptive string as diagnostic payload.
  - If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the server **MAY** create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the authentication credential associated with the Recipient ID to be stored in the new Recipient Context. Such a configuration is application-specific. If the application does not specify dynamic derivation of new Recipient Contexts, then the server **SHALL** stop processing the request.
- In Step 4, the AAD is modified as described in Section 3.4 of this document.
- In Step 6, the server also verifies the countersignature by using the public key from the client's authentication credential stored in the associated Recipient Context. In particular:
  - If the server does not have the public key of the client yet, the server **MUST** stop processing the request and **MAY** respond with a 5.03 (Service Unavailable) response. The response **MAY** include a Max-Age Option, indicating to the client the number of seconds after which to retry. If the Max-Age Option is not present, Section 5.10.5 of [RFC7252] specifies a default retry time of 60 seconds.
  - The signature verification as defined below **SHOULD** be performed before decrypting the COSE object. An exception applies to implementations that cannot perform the two steps in this order. Those implementations **MUST** ensure that no access to the plaintext is possible before a successful signature verification and **MUST** prevent any possible leak of time-related information that can yield side-channel attacks.
  - The server retrieves the encrypted countersignature ENC_SIGNATURE from the message payload and computes the original countersignature SIGNATURE as:

    SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 4.2.
  - The server verifies the original countersignature SIGNATURE as described in Sections 3.2 and 3.3 of [RFC9338] by using the client's public key and according to the Signature Algorithm in the Security Context.

In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 3.4 of this document, and the ciphertext of the COSE object as payload.

- If the signature verification fails, the server **SHALL** stop processing the request, **SHALL NOT** update the Replay Window, and **MAY** respond with a 4.00 (Bad Request) response. Such a response **MAY** include an Outer Max-Age Option with value zero, and its diagnostic payload **MAY** contain a string, which, if present, **MUST** be "Decryption failed" as if the decryption of the COSE object had failed.
- When decrypting the COSE object using the Recipient Key, the Group Encryption Algorithm from the Common Context **MUST** be used.

- Additionally, if the used Recipient Context was created upon receiving this request and the message is not verified successfully, the server **MAY** delete that Recipient Context. When this behavior is specified by the application, it mitigates attacks that aim at overloading the server's storage.

  If the server deletes the used Recipient Context in this particular circumstance, then this deletion does not require the server to initialize the Replay Window of any new Recipient Context created later within the Security Context as invalid (see Section 2.6.1.2).

A server **SHOULD NOT** process a request if the received Recipient ID ('kid') is equal to its own Sender ID in its own Sender Context. However, in some applications, the server can prepare a request to be sent to itself (e.g., see [CoAP-MULT-NOTIF]), in which case such requests would be expected.

In addition, the following applies if the request establishes a long exchange and the server intends to reply with multiple responses:

- The server **MUST** store the value of the 'kid' parameter from the request and retain it until the last response has been sent. The server **MUST NOT** update the stored value of the 'kid' parameter associated with the request, even if the client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.6.3.1).
- The server **MUST** store the value of the 'kid context' parameter from the request and retain it until the last response has been sent, i.e., beyond a possible change of ID Context following a group rekeying (see Section 12.2). That is, upon establishing a new Security Context with a new Gid as ID Context (see Section 2.6.3.2), the server **MUST NOT** update the stored value of a 'kid context' parameter associated with the request.

7.3. Protecting the Response

When using the group mode to protect a response, a server proceeds as described in Section 8.3 of [RFC8613] with the following modifications.

Note that the server always protects a response with the Sender Context from its latest Security Context, and that establishing a new Security Context resets the Sender Sequence Number to 0 (see Sections 2.6.3.1 and 2.6.3.2).

- In Step 2, the AAD is modified as described in Section 3.4 of this document.

  In addition, the following applies if the server intends to reply with multiple responses within the long exchange established by the corresponding request:

  - The server **MUST** use the stored value of the 'kid' parameter from the request (see Section 7.2) as the value for the 'request_kid' parameter in the external_aad (see Section 3.4).
  - The server **MUST** use the stored value of the 'kid context' parameter from the request (see Section 7.2) as the value for the 'request_kid_context' parameter in the external_aad (see Section 3.4).

- In Step 3, if either of the following conditions applies, the server **MUST** include its Sender Sequence Number as Partial IV in the response and use it to build the nonce to protect the response. This prevents the server from reusing the nonce from the request together with the same encryption key.

  - The response is not the first response that the server sends to the request.
  - The server is using a different Security Context for the response than the one that was used to verify the request (see Section 12.2).

- In Step 4, the encryption of the COSE object is modified as described in Section 3 of this document. The encoding of the compressed COSE object is modified as described in Section 4 of this document. In particular, the Group Flag **MUST** be set to 1. The Group Encryption Algorithm from the Common Context **MUST** be used.

  In addition, the following applies:

  - If the server is using a different ID Context (Gid) for the response than the one that was used to verify the request (see Section 12.2) and this is the first response from the server to that request, then the new ID Context **MUST** be included in the 'kid context' parameter of the response.
  - The server may be replying to a request that was protected with an old Security Context. After completing the establishment of a new Security Context, the server **MUST** protect all the responses to that request with the Sender Context of the new Security Context.

    For each ongoing long exchange, the server can help the client to synchronize by also including the 'kid context' parameter in responses following a group rekeying, with the value set to the ID Context (Gid) of the new Security Context.

    If there is a known upper limit to the duration of a group rekeying, the server **SHOULD** include the 'kid context' parameter during that time. Otherwise, the server **SHOULD** include it until the Max-Age has expired for the last response sent before the installation of the new Security Context.

- The server can obtain a new Sender ID from the Group Manager when individually rekeyed (see Section 2.6.3.1) or when re-joining the group. In such a case, the server can help the client to synchronize by including the 'kid' parameter in a response protected in group mode even when the request was protected in pairwise mode (see Section 8.3).

  That is, when responding to a request protected in pairwise mode, the server **SHOULD** include the 'kid' parameter in a response protected in group mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

- In Step 5, the countersignature is computed and the format of the OSCORE message is modified as described in Sections 3 and 4 of this document. In particular the payload of the Group OSCORE message includes also the encrypted countersignature (see Section 4.1).

7.4. Verifying the Response

Upon receiving a protected response with the Group Flag set to 1, following the procedure in Section 6, a client proceeds as described in Section 8.4 of [RFC8613] with the following modifications.

Note that a client may receive a response protected with a Security Context different from the one used to protect the corresponding request, and that, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Section 2.6.3.2).

- In Step 2, the decoding of the compressed COSE object is modified as described in Section 4 of this document. In particular, a 'kid' may not be present if the response is a reply to a request protected in pairwise mode. In such a case, the client assumes the response 'kid' to be the Recipient ID for the server for which the request protected in pairwise mode was intended.

  If the response 'kid context' matches an existing ID Context (Gid) but the received/assumed 'kid' does not match any Recipient ID in this Security Context, then the client **MAY** create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the authentication credential associated with the Recipient ID to be stored in the new Recipient Context. If the application does not specify dynamic derivation of new Recipient Contexts, then the client **SHALL** stop processing the response.

- In Step 3, the AAD is modified as described in Section 3.4 of this document.

  In addition, the following applies if the client processes a response to a request within a long exchange:

  - The client **MUST** use the stored value of the 'kid' parameter from the request (see Section 7.1) as the value for the 'request_kid' parameter in the external_aad (see Section 3.4).
  - The client **MUST** use the stored value of the 'kid context' parameter from the request (see Section 7.1) as the value for the 'request_kid_context' parameter in the external_aad (see Section 3.4).

This ensures that, throughout a long exchange, the client can correctly verify the received responses, even if the client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.6.3.1), as well as when it installs a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 2.6.3.2).

- In Step 5, the client also verifies the countersignature by using the public key from the server's authentication credential stored in the associated Recipient Context. In particular:
  - The signature verification as defined below **SHOULD** be performed before decrypting the COSE object. An exception applies to implementations that cannot perform the two steps in this order. Those implementations **MUST** ensure that no access to the plaintext is possible before a successful signature verification and **MUST** prevent any possible leak of time-related information that can yield side-channel attacks.
  - The client retrieves the encrypted countersignature ENC_SIGNATURE from the message payload and computes the original countersignature SIGNATURE as:

    SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 4.2.

    The client verifies the original countersignature SIGNATURE. This is analogous to what is defined in Section 7.2 for the signature verification of a request, with the client using the server's public key and referring to the present response and the COSE object conveyed therein.
  - If the signature verification fails, the client **SHALL** stop processing the response and **SHALL NOT** update the Response Number associated with the server.
  - After a successful signature verification, the client also performs the following actions if the request was protected in pairwise mode (see Section 8.3):
    - If the 'kid' parameter is present in the response, the client checks whether this received 'kid' is equal to the expected 'kid', i.e., the known Recipient ID for the server for which the request was intended.
    - If the 'kid' parameter is not present in the response, the client checks whether the server that has sent the response is the same one for which the request was intended. This can be done by checking that the public key used to verify the countersignature of the response is equal to the public key included in the authentication credential Recipient Auth Cred, which was taken as input to derive the Pairwise Sender Key used for protecting the request (see Section 2.5.1).

    In either case, if the client determines that the response has come from a different server than the expected one, then the client: i) **SHALL** discard the response and **SHALL NOT** deliver it to the application, and ii) **SHALL NOT** update the Response Number associated with the server.

    Otherwise, the client hereafter considers the received 'kid' as the current Recipient ID for the server.

- In Step 5, when decrypting the COSE object using the Recipient Key, the Group Encryption Algorithm from the Common Context **MUST** be used.

In addition, the client performs the following actions if the response is received within a long exchange.

- The ordering and the replay protection of responses received from the server during the long exchange are performed as per Section 5.3.1 of this document by using the Response Number associated with that server within that long exchange. In the case of unsuccessful decryption and verification of a response, the client **SHALL NOT** update the Response Number associated with the server.
- When receiving the first valid response from the server within the long exchange, the client **MUST** store the kid "kid1" of that server for that long exchange. If the 'kid' field is included in the OSCORE Option of the response, its value specifies "kid1". If the request was protected in pairwise mode (see Section 8.3), the 'kid' field may not be present in the OSCORE Option of the response (see Section 3.2). In this case, the client assumes "kid1" to be the Recipient ID for the server for which the request was intended.
- When receiving another valid response to the same request from the same server -- which can be identified and recognized through the same public key used to verify the countersignature and included in the server's authentication credential -- the client determines the kid "kid2" of the server as above for "kid1", and it **MUST** check whether "kid2" is equal to the stored "kid1".

  If "kid1" and "kid2" are different, the client **SHOULD NOT** accept the response as valid to be delivered to the application and **SHOULD NOT** update the Response Number associated with the server. Exceptions can apply as the client can retain the information required to order the responses, or if the client application does not require response ordering altogether. Servers **MUST NOT** rely on clients tolerating this unless it was explicitly agreed on (e.g., as part of the group's setup).

Note that, if "kid2" is different from "kid1" and the 'kid' field is omitted from the response -- which is possible if the request was protected in pairwise mode -- then the client will compute a wrong keystream to decrypt the countersignature (i.e., by using "kid1" rather than "kid2" in the 'id' field of the 'info' array in Section 4.2), thus subsequently failing to verify the countersignature and discarding the response.

This ensures that the client remains able to correctly perform the ordering and replay protection of responses within a long exchange, even if the server legitimately starts using a new Sender ID, as received from the Group Manager when individually rekeyed (see Section 2.6.3.1) or when re-joining the group.

- In Step 8, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client **MAY** delete that Recipient Context. When this behavior is specified by the application, it mitigates attacks that aim at overloading the client's storage.

  If the client deletes the used Recipient Context in this particular circumstance, then this deletion does not require the client to initialize the Replay Window of any new Recipient Context created later within the Security Context as invalid (see Section 2.6.1.2).

7.5. External Signature Checkers

When a message is protected in group mode, it is possible for designated external signature checkers to verify the countersignature of the message. For example, an external signature checker can be an intermediary gateway that intercepts messages protected in group mode and ensures that they reach the intended recipients only if it successfully verifies their countersignatures.

Since they do not join a group as members, external signature checkers need to retrieve the authentication credentials and other information (e.g., the Sender IDs) associated with group members from the Group Manager, together with other selected group data such as the current Signature Encryption Key (see Section 2.1.9). Section 12.3 describes how the Group Manager supports signature checkers.

When receiving a message protected in group mode, a signature checker proceeds as follows.

- The signature checker retrieves the encrypted countersignature ENC_SIGNATURE from the message payload and computes the original countersignature SIGNATURE as:

  SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

  where KEYSTREAM is derived as per Section 4.2.

- The signature checker verifies the original countersignature SIGNATURE according to what is defined in Section 7.2 for requests and in Section 7.4 for responses by using the public key of the sender endpoint as included in that endpoint's authentication credential. The signature checker determines the right authentication credential based on the ID Context (Gid) and the Sender ID of the sender endpoint.

Note that the following applies when attempting to verify the countersignature of a response message:

- The response may not include a Partial IV. In such a case, the signature checker considers the same value of the Partial IV from the corresponding request, i.e., the request matching with the response by CoAP Token value.
- The response may not include an ID Context. In such a case, the signature checker considers the current ID Context used in the group. By default, such ID Context has the same value of the ID Context from the corresponding request, i.e., the request matching with the response by CoAP Token value.

  Since the ID Context (Gid) changes every time the group is rekeyed (see Section 12.2), the server that originated the response could have used a different ID Context when protecting the response compared to the ID Context that it used to verify (that the client used to protect) the request matching with the response (see Section 7.3). In particular, Section 7.3 defines the rules for the inclusion of the ID Context in responses protected in group mode, e.g., following a group rekeying.

When the signature checker receives a response protected in group mode that includes an ID Context different from the one known to be current, the signature checker takes that as an indication that a group rekeying has occurred. Then, the signature checker retrieves updated group data from the Group Manager, such as the current ID Context and Signature Encryption Key used in the group (see Section 12.3). From then on, the signature checker considers the ID Context retrieved from the Group Manager as the current ID Context used in the group.

- The response may not include a Sender ID. This can happen when the response protected in group mode matches a request protected in pairwise mode (see Sections 3.2 and 8.3) with a case in point provided by [CACHEABLE-OSCORE]. In such a case, the signature checker needs to use other means (e.g., source addressing information of the server endpoint) to identify the correct authentication credential including the public key to use for verifying the countersignature of the response.

The particular actions following a successful or unsuccessful verification of the countersignature are application-specific and out of the scope of this document.

8. Message Processing in Pairwise Mode

When using the pairwise mode of Group OSCORE, messages are protected and processed as in [RFC8613] with the modifications described in this section. The security objectives of the pairwise mode are discussed in Appendix A.2.

The possible use of the pairwise mode is indicated by the Group Manager as part of the group data provided to new group members when joining the group, according to which the parameters AEAD Algorithm and Pairwise Key Agreement Algorithm in the Security Context are set (see Section 2).

The pairwise mode takes advantage of an existing Security Context to establish keying material shared exclusively with each other member. For encryption and decryption operations in pairwise mode, the AEAD Algorithm from the Common Context is used (see Section 2.1.1).

In order to use the pairwise mode in a group where the group mode is also used (i.e., Group Encryption Algorithm and Signature Algorithm in the Security Context are set), the public/private key pairs used for signature operations of the group mode **MUST** be possible to also use for key agreement operations. For example, this can rely on signing operations using ECDSA and encryption operations using AES Counter Mode with CBC-MAC (AES-CCM) with keying material derived through ECDH.

The pairwise mode does not support external verifiers of source authentication and message integrity like the group mode does, e.g., for external signature checkers (see Section 7.5).

An endpoint implementing only a silent server does not support the pairwise mode.

Endpoints using the CoAP Echo Option [RFC9175] in a group where the AEAD Algorithm and Pairwise Key Agreement Algorithm are set **MUST** support the pairwise mode. When using the challenge-response method defined in Section 9, this prevents the attack described in Section 13.9, which leverages requests sent over unicast to a single group member and protected in group mode.

The pairwise mode cannot be used to protect messages intended for multiple recipients, as the keying material used for the pairwise mode is shared only between two endpoints.

However, a sender can use the pairwise mode to protect a message sent to (but not intended for) multiple recipients, if interested in a response from only one of them. For instance, this is useful to support the address discovery service defined in Section 8.1 when a single 'kid' value is indicated in the payload of a request sent to multiple recipients, e.g., over multicast.

8.1. Preconditions

In order to protect an outgoing message in pairwise mode, the sender needs to know the authentication credential and the Recipient ID for the recipient endpoint as stored in the Recipient Context associated with that endpoint (see Section 2.5.4).

Typically, the sender endpoint sends the message protected in pairwise mode over unicast, so that the message is delivered only to the intended recipient endpoint for which it is protected. This requires the sender to know the individual address of that recipient endpoint, which the sender may not know at any given point in time. For instance, right after having joined the group, a client may know the authentication credential and Recipient ID for a given server, but not the addressing information required to reach it with an individual, one-to-one request.

In order to make addressing information of individual endpoints available, servers in the group **MAY** expose a resource to which a client can send a request targeting a set of servers, identified by their 'kid' values specified in the request payload, or implicitly if the request is sent in pairwise mode. Further details of such an interface are out of scope for this document.

8.2. Main Differences from OSCORE

The pairwise mode protects messages between two members of a group, essentially following [RFC8613] but with the following notable differences:

- The 'kid' and 'kid context' parameters of the COSE object are used as defined in Section 3.2 of this document.
- The external_aad defined in Section 3.4 of this document is used for the encryption and decryption process.
- The Pairwise Sender/Recipient Keys defined in Section 2.5 of this document are used for the encryption and decryption process, instead of the Sender Keys and Recipient Keys.

8.3. Protecting the Request

When using the pairwise mode to protect a request, a client **SHALL** proceed as described in Section 8.1 of [RFC8613] with the differences summarized in Section 8.2 of this document.

Furthermore, when sending a request that establishes a long exchange, what is specified in Section 7.1 of this document applies, with respect to storing the value of the 'kid' and 'kid context' parameters and to storing an invariant identifier of the group.

8.4. Verifying the Request

Upon receiving a protected request with the Group Flag set to 0, following the procedure in Section 6, a server **SHALL** proceed as described in Section 8.2 of [RFC8613] with the differences summarized in Section 8.2 of this document. The following differences also apply:

- If the server discards the request due to not retrieving a Security Context associated with the OSCORE group or to not supporting the pairwise mode, the server **MAY** respond with a 4.01 (Unauthorized) error message or a 4.02 (Bad Option) error message, respectively. When doing so, the server **MAY** set an Outer Max-Age Option with value zero and **MAY** include a descriptive string as diagnostic payload.
- If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.5.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- What is specified in Section 7.2 of this document applies with respect to the following points:
  - The possible, dynamic creation and configuration of a Recipient Context upon receiving the request.
  - The possible deletion of a Recipient Context created upon receiving the request, in the case that the request is not verified successfully.
  - The rule about processing the request where the received Recipient ID ('kid') is equal to the server's Sender ID.
  - The storing of the value of the 'kid' and 'kid context' parameters from the request if the server intends to reply with multiple responses within the long exchange established by the request.

8.5. Protecting the Response

When using the pairwise mode to protect a response, a server **SHALL** proceed as described in Section 8.3 of [RFC8613] with the differences summarized in Section 8.2 of this document. The following differences also apply:

- What is specified in Section 7.3 of this document applies with respect to the following points.
  - The protection of a response when using a different Security Context than the one used to verify the corresponding request (see Section 12.2). That is, the server always protects a response with the Sender Context from its latest Security Context, and establishing a new Security Context resets the Sender Sequence Number to 0 (see Sections 2.6.3.1 and 2.6.3.2).
  - The use of the stored value of the 'kid' and 'kid context' parameters, if the server intends to reply with multiple responses within the long exchange established by the request.
  - The rules for the inclusion of the server's Sender Sequence Number as Partial IV in a response, as used to build the nonce to protect the response.
  - The rules for the inclusion of the ID Context (Gid) in the 'kid context' parameter of a response, if the ID Context used for the response differs from the one used to verify the request (see Section 12.2), also for helping the client to synchronize.
  - The rules for the inclusion of the Sender ID in the 'kid' parameter of a response to a request that was protected in pairwise mode, if the server has obtained a new Sender ID from the Group Manager when individually rekeyed (see Section 2.6.3.1), thus helping the client to synchronize.

8.6. Verifying the Response

Upon receiving a protected response with the Group Flag set to 0, following the procedure in Section 6, a client **SHALL** proceed as described in Section 8.4 of [RFC8613] with the differences summarized in Section 8.2 of this document. The following differences also apply:

- The client may receive a response protected with a Security Context different from the one used to protect the corresponding request. Also, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Sections 2.2 and 2.6.3).
- The same as described in Section 7.4 applies with respect to handling the 'kid' parameter of the response, when received as a reply to a request protected in pairwise mode. In this case too, the client can check whether the replying server is the expected one by relying on the server's public key. However, since the response is protected in pairwise mode, the public key is not used for verifying a countersignature as in Section 7.4. Instead, the expected server's authentication credential -- namely Recipient Auth Cred and including the server's public key -- was taken as input to derive the Pairwise Recipient Key used to decrypt and verify the response (see Section 2.5.1).
- If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.5.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

- What is specified in Section 7.4 of this document applies with respect to the following points.
  - The possible, dynamic creation and configuration of a Recipient Context upon receiving the response.
  - The use of the stored value of the 'kid' and 'kid context' parameters when processing a response received within a long exchange.
  - The performing of ordering and replay protection for responses received within a long exchange.
  - The possible deletion of a Recipient Context created upon receiving the response in the case that the response is not verified successfully.

9. Challenge-Response-Based Freshness and Replay Window Recovery

This section describes how a server endpoint can verify freshness of a request by means of a challenge-response exchange with a client using the Echo Option for CoAP specified in Section 2 of [RFC9175]. The same mechanism, with small alterations, is also used by the server when first processing a request using a Recipient Context whose Replay Window was initialized as invalid.

If the application requires freshness, e.g., according to time- or event-based policies (see Section 2.5.1 of [RFC9175]), a server proceeds as described below upon receiving a request from a particular client for the first time.

The server processes the message as described in this document but does not deliver it to the application, even if the message is valid. Instead, the server replies to the client with a Group OSCORE-protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The server **MUST** use its Partial IV when generating the nonce for protecting the response conveying the Echo Option and **MUST** include the Partial IV in the response.

The Echo Option value **SHOULD NOT** be reused; if it is reused, it **MUST** be highly unlikely to have been recently used with this client. Since this response is protected with the Security Context used in the group, the client will consider the response valid upon successfully decrypting and verifying it.

The server stores the Echo Option value included in the response together with the pair (gid,kid), where 'gid' is the Group Identifier of the OSCORE group and 'kid' is the Sender ID of the client in the group. These are specified in the 'kid context' and 'kid' fields of the OSCORE Option of the request, respectively. After a group rekeying has been completed and a new Security Context has been established in the group, which also results in a new Group Identifier (see Section 12.2), the server **MUST** delete all the stored Echo values associated with members of the group.

After receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, a subsequent client request sent to that server and echoing the Echo Option value **MUST** be a message sent unicast to that server.

In the group, if the AEAD Algorithm and Pairwise Key Agreement Algorithm are set in the Security Context, the client **MUST** use the pairwise mode to protect the request as per Section 8.3. Note that, as defined in Section 8, endpoints that are members of such a group and that use the Echo Option support the pairwise mode. In a group where the AEAD Algorithm and Pairwise Key Agreement Algorithm are not set, only the group mode can be used. Hence, requests including the Echo Option can be protected only with the group mode, with the caveat due to the risk for those requests to be redirected to a different server than the intended one as discussed in Section 13.9.

The client does not necessarily resend the same request, but can instead send a more recent one if the application permits it. This allows the client to not retain previously sent requests for full retransmission unless the application explicitly requires otherwise. In either case, the client uses a fresh Sender Sequence Number value from its own Sender Context. If the client stores requests for possible retransmission with the Echo Option, it should not store a given request for longer than a preconfigured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the OSCORE Option as part of the COSE object (see Section 3).

Upon receiving the unicast request including the Echo Option, the server performs the following verifications:

- If the server does not store an Echo Option value for the pair (gid,kid), it considers i) the time t1 when it has established the Security Context used to protect the received request, and ii) the time t2 when the request has been received. Since a valid request cannot be older than the Security Context used to protect it, the server verifies that (t2 - t1) is less than the largest amount of time acceptable to consider the request fresh.
- If the server stores an Echo Option value for the pair (gid,kid) associated with that same client in the same group, the server verifies that the option value equals that same stored value previously sent to that client.

If the verifications above fail, the server **MUST NOT** process the request further and **MAY** send a 4.01 (Unauthorized) response including an Echo Option, hence performing a new challenge-response exchange.

If the verifications above are successful, the server considers the Recipient Context associated with the sender client and proceeds as follows:

- If the Replay Window is invalid, the steps below occur:
  1. The server updates the Replay Window by marking as received the Sender Sequence Number from the latest received request. This becomes the lower limit of the Replay Window while all the greater Sender Sequence Number values within the Replay Window are marked as not received.
  2. The server makes the Replay Window valid and accepts the request as fresh.
- If the Replay Window is already valid, the server discards the verification result and accepts the request as fresh or treats it as a replay according to the existing Replay Window.

A server should not deliver requests from a given client to the application until one valid request from that same client has been verified as fresh via an echoed Echo Option included therein. A server may perform the challenge-response described above at any time, e.g., after a device reboot occurred in an unprepared way. A client has to be ready to perform the challenge-response based on the Echo Option if a server starts it.

Message freshness is further discussed in Section 13.14.

10. Implementation Compliance

Like in [RFC8613], HKDF SHA-256 is the mandatory-to-implement HKDF.

An endpoint may support only the group mode, or only the pairwise mode, or both.

For endpoints that support the group mode, the following applies:

- For endpoints that use authenticated encryption, the AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [RFC9053] is mandatory to implement as Group Encryption Algorithm (see Section 2.1.7).
- For endpoints that use non-authenticated encryption, the algorithm A128CTR defined in Section 4 of [RFC9459] is mandatory to implement as Group Encryption Algorithm (see Section 2.1.7).
- Section 6 of [RFC9459] mandates that COSE libraries supporting the AES-CTR algorithm and accepting AAD as input must return an error if AAD is provided when such a non-AEAD content encryption algorithm is selected.

  If the used Group Encryption Algorithm (see Section 2.1.7) does not provide integrity protection, the following applies:

  When invoking the execution of the Group Encryption Algorithm, the Group OSCORE implementation **MUST NOT** provide any AAD to the COSE library, unless AAD is always expected as input. In the latter case, the AAD will not be protected by the Group Encryption Algorithm, which is unable to do so.

  If the used COSE library adheres to the mandate in Section 6 of [RFC9459], then a Group OSCORE implementation requires that the COSE library supports using the Group Encryption Algorithm without taking AAD as input.
- For many constrained Internet of Things (IoT) devices, it is problematic to support more than one signature algorithm. The following applies with respect to the Signature Algorithm (see Section 2.1.8).

  Less constrained endpoints **MUST** implement at least one of the following and **SHOULD** implement both: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032] and/or the ECDSA signature algorithm together with the elliptic curve P-256.

  Constrained endpoints **MUST** implement at least one of the following and, if affordable, **SHOULD** implement both: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032] and/or the ECDSA signature algorithm together with the elliptic curve P-256.

- Endpoints that implement the ECDSA signature algorithm **MAY** use "deterministic ECDSA" as specified in [RFC6979]. Pure deterministic elliptic-curve signature algorithms such as deterministic ECDSA and EdDSA have the advantage of not requiring access to a source of high-quality randomness. However, these signature algorithms have been shown vulnerable to some side-channel and fault injection attacks due to their determinism, which can result in extracting a device's private key. As suggested in Section 2.1.1 of [RFC9053], this can be addressed by combining both randomness and determinism [SIGS-WITH-NOISE].

For endpoints that support the pairwise mode, the following applies:

- The AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [RFC9053] is mandatory to implement as AEAD Algorithm (see Section 2.1.1).
- The ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [RFC9053] is mandatory to implement as Pairwise Key Agreement Algorithm (see Section 2.1.10).
- The following applies with respect to ECDH curves.

  Less constrained endpoints **MUST** implement at least one of the following ECDH curves and **SHOULD** implement both: the X25519 curve [RFC7748] and/or the P-256 curve.

  Constrained endpoints **MUST** implement at least one of the following ECDH curves and, if affordable, **SHOULD** implement both: the X25519 curve [RFC7748] and/or the P-256 curve.

Constrained IoT devices may alternatively represent Montgomery curves and (twisted) Edwards curves [RFC7748] in the short-Weierstrass form Wei25519, with which the algorithms ECDSA25519 and ECDH25519 can be used for signature operations and Diffie-Hellman secret calculation, respectively [CURVE-REPR].

11. Web Linking

The use of Group OSCORE or OSCORE [RFC8613] **MAY** be indicated by a target "gosc" attribute in a web link [RFC8288] to a resource, e.g., using a CoRE link-format document [RFC6690] if the resource is accessible over CoAP.

The "gosc" attribute is a hint indicating that the destination of that link is only accessible using Group OSCORE or OSCORE, and unprotected access to it is not supported. Note that this is simply a hint; it does not include any security context material or any other information required to run Group OSCORE or OSCORE.

A value **MUST NOT** be given for the "gosc" attribute; any present value **MUST** be ignored by parsers. The "gosc" attribute **MUST NOT** appear more than once in a given link-value; occurrences after the first **MUST** be ignored by parsers.

When a link-value includes the "gosc" attribute, the link-value **MUST** also include the "osc" attribute defined in Section 9 of [RFC8613]. If the endpoint parsing the link-value supports Group OSCORE and understands the "gosc" attribute, then the parser **MUST** ignore the "osc" attribute, which is overshadowed by the "gosc" attribute.

The example in Figure 3 shows a use of the "gosc" attribute: the client does resource discovery on a server and gets back a list of resources, one of which includes the "gosc" attribute indicating that the resource is protected with Group OSCORE or OSCORE. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core

RES: 2.05 Content
    </sensors/temp>;gosc;osc,
    </sensors/light>;if="sensor"
```

Figure 3: Example of Using the "gosc" Attribute in a Web Link

12. The Group Manager

As with OSCORE, endpoints communicating with Group OSCORE need to establish the relevant Security Context. Group OSCORE endpoints need to acquire OSCORE input parameters, information about the group(s), and information about other endpoints in the group(s).

Every group is associated with a Group Manager that is responsible for distributing security parameters and keying material within the group, among other tasks. The details of how the Group Manager interacts with (candidate) group members or with external entities like signature checkers, as well as the protocols used for those interactions, are out of scope.

The Group Manager assigns unique Group Identifiers (Gids) to the groups under its control. Within each of such groups, the Group Manager assigns unique Sender IDs (and thus Recipient IDs) to the respective group members. The maximum length of Sender IDs depends on the length of the nonce for the algorithms used in the group (see Section 2.2).

The Gid value assigned to a group is associated with a dedicated space for the values of Sender ID and Recipient ID of the members of that group. When an endpoint (re-)joins a group, it is provided with the current Gid to use in the group. The Group Manager also assigns an integer Key Generation Number counter to each of its groups, identifying the current version of the keying material used in that group. Further details about identifiers and keys are provided in Section 12.2.

The Group Manager maintains records of the authentication credentials of endpoints in a group, and provides members of the group with information about the group and other group members (see Section 12.1). Optionally, the Group Manager provides information about the group and its members to external entities with a specific role, such as signature checkers (see Section 12.3).

The list of responsibilities of the Group Manager is compiled in Appendix D.

One realization of Group Manager is specified in [ACE-KEY-GROUP-OSCORE], where the process by which an endpoint (re-)joins a group is based on the Authentication and Authorization for Constrained Environments (ACE) framework [RFC9200].

12.1. Setup of New Endpoints

From the Group Manager, an endpoint acquires group data such as the Gid and OSCORE input parameters including its own Sender ID, with which it can derive the Sender Context.

When joining the group or later on as a group member, an endpoint can also retrieve from the Group Manager the authentication credential of the Group Manager as well as the authentication credentials and other information (e.g., the Sender IDs) associated with other members of the group, with which it can derive the corresponding Recipient Context. An application can configure a group member to asynchronously retrieve information about Recipient Contexts, e.g., by observing [RFC7641] a resource at the Group Manager to get updates on the group membership.

When endpoints join the group, the Group Manager collects the endpoints' authentication credentials and **MUST** verify proof of possession of the respective private key. As an example, such proof of possession is possible to achieve during the join process provided by the realization of Group Manager specified in [ACE-KEY-GROUP-OSCORE]. Together with the requested authentication credentials of other group members, the Group Manager **MUST** provide the joining endpoints with the Sender ID of the associated group members and the current Key Generation Number in the group (see Section 12.2).

An endpoint may join a group, for example, by explicitly interacting with the responsible Group Manager or by being configured with some tool performing the tasks of the Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

Communications that the Group Manager has with joining endpoints and group members **MUST** be secured. Specific details on how to secure such communications are out of the scope of this document.

The Group Manager **MUST** verify that the joining endpoint is authorized to join the group. To this end, the Group Manager can directly authorize the joining endpoint or expect it to provide authorization evidence previously obtained from a trusted entity. Further details about the authorization of joining endpoints are out of the scope of this document.

In the case of successful authorization check, the Group Manager provides the joining endpoint with the keying material to initialize the Security Context. The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

12.2. Management of Group Keying Material

In order to establish a new Security Context for a group, the Group Manager **MUST** generate and assign to the group a new Group Identifier (Gid) and a new value for the Master Secret parameter. When doing so, a new value for the Master Salt parameter **MAY** also be generated and assigned to the group. When establishing the new Security Context, the Group Manager **SHOULD**

preserve the current value of the Sender ID of each group member in order to ensure an efficient key rollover. Exceptions can apply if there are compelling reasons for making available again some of the Sender ID values currently used.

The specific group key management scheme used to distribute new keying material is out of the scope of this document. A simple group key management scheme is defined in [ACE-KEY-GROUP-OSCORE]. Different group key management schemes rely on different approaches to compose and deliver rekeying messages, i.e., individually targeting single recipients, or targeting multiple recipients at once (e.g., over UDP/IP multicast), or a combination of the two approaches. As long as it is viable for the specific rekeying message to be delivered and it is supported by the intended message recipient(s), using a reliable transport to deliver a rekeying message should be preferred, as it reduces chances of group members missing a rekeying instance.

Irrespective of the transport used being reliable or unreliable, appropriate congestion control **MUST** be enforced. If the key distribution traffic uses CoAP over UDP or over other unreliable transports, mechanisms for enforcing congestion control are specified in Section 4.7 of [RFC7252] and in Section 3.6 of [RFC10020] for the case of group communication (e.g., over UDP/IP multicast). If, irrespective of using CoAP, the key distribution traffic relies on alternative setups with unreliable transports, one can rely on general congestion-control mechanisms such as the Datagram Congestion Control Protocol (DCCP) [RFC4340] or on dedicated congestion-control mechanisms for the transport specifically used (e.g., those defined in [RFC9002] for QUIC [RFC9000]).

The set of group members should not be assumed as fixed, i.e., the group membership is subject to changes, possibly on a frequent basis.

The Group Manager **MUST** rekey the group without undue delay when one or more endpoints leave the group. An endpoint may leave the group on its own initiative or may be evicted from the group by the Group Manager, e.g., when the endpoint is compromised or is suspected to be compromised (as determined by the Group Manager through its own means or based on information that it obtains from a trusted source, such as an Intrusion Detection System or an issuer of authentication credentials). In either case, rekeying the group excludes such endpoints from future communications in the group, and thus preserves forward security. If a network node is compromised or suspected to be compromised, the Group Manager **MUST** evict from the group all the endpoints hosted by that node that are members of the group and rekey the group accordingly.

If required by the application, the Group Manager **MUST** also rekey the group when one or more new joining endpoints are added to the group, thus preserving backward security.

The Group Manager **MAY** also rekey the group for other reasons, e.g., according to an application-specific rekeying period or scheduling.

Separately for each group, the value of the Key Generation Number increases by one each time the Group Manager distributes new keying material to that group (see below).

The establishment of the new Security Context for the group takes the following steps.

1. The Group Manager **MUST** increment the Key Generation Number for the group by 1. It is up to the Group Manager what actions to take when a wrap-around of the Key Generation Number is detected.
2. The Group Manager **MUST** build a set of stale Sender IDs, including:
   - The Sender IDs that have been relinquished since the latest assignment of the current Gid to the group (see Section 2.6.3.1).
   - The current Sender IDs of the group members that the upcoming group rekeying aims to exclude from future group communications, if any.
3. The Group Manager rekeys the group by distributing:
   - The new keying material, i.e., the new Master Secret, the new Gid, and (optionally) the new Master Salt.
   - The new Key Generation Number from Step 1.
   - The set of stale Sender IDs from Step 2.

   Further information may be distributed depending on the specific group key management scheme used in the group.

When receiving the new group keying material, a group member considers the received stale Sender IDs and performs the following actions:

- The group member **MUST** remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group.
- The group member **MUST** delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

After that, the group member installs the new keying material and derives the corresponding new Security Context.

A group member might miss one or more consecutive instances of group rekeying. As a result, the group member will retain old group keying material with Key Generation Number GEN_OLD. Eventually, the group member can notice the discrepancy, e.g., by repeatedly failing to verify incoming messages or by explicitly querying the Group Manager for the current Key Generation Number. Once the group member gains knowledge of having missed a group rekeying, it **MUST** delete the old keying material it stores.

Then, the group member proceeds according to the following steps.

1. The group member retrieves from the Group Manager the current group keying material together with the current Key Generation Number GEN_NEW. The group member **MUST NOT** install the obtained group keying material yet.
2. The group member asks the Group Manager for the set of stale Sender IDs between GEN_OLD and GEN_NEW.

3. If no exact and complete indication can be obtained from the Group Manager, the group member **MUST** remove all the authentication credentials from its list of group members' authentication credentials used in the group and **MUST** delete all its Recipient Contexts used in the group.

   Otherwise, the group member **MUST** remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group and **MUST** delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

4. The group member installs the current group keying material and derives the corresponding new Security Context.

Alternatively, the group member can re-join the group. In such a case, the group member **MUST** take one of the following two actions:

- First, the group member performs Steps 2 and 3 above. Then, the group member re-joins the group.
- The group member re-joins the group with the same roles it currently has in the group and, during the re-join process, asks the Group Manager for the authentication credentials of all the current group members.

  Then, given Z (the set of authentication credentials received from the Group Manager), the group member removes every authentication credential that is not present in Z from its list of group members' authentication credentials used in the group. Furthermore, it deletes each of its Recipient Contexts used in the group that does not include any of the authentication credentials found in Z.

By removing authentication credentials and deleting Recipient Contexts associated with stale Sender IDs, it is ensured that a recipient endpoint storing the latest group keying material does not store the authentication credentials of sender endpoints that are not current group members. This, in turn, allows group members to rely on stored authentication credentials to confidently verify the group membership of sender endpoints when receiving incoming messages protected in group mode (see Section 7).

Strictness in managing the authentication credentials and Recipient Contexts associated with other group members is required for two reasons. First, as further discussed in Section 13.1, it ensures that the group mode can be used securely, even in a group where the Group Encryption Algorithm does not provide integrity protection (see Section 2.1.7) and external signature checkers are used (see Section 7.5). Second, it ensures that the wrong (old) authentication credential associated with a group member A is never used with a Sender ID that used to be associated with A and has been later issued to a different group member B (see Section 12.2.1.2), thus preventing the need to recover from an identity mix-up.

12.2.1. Recycling of Identifiers

This section specifies how the Group Manager handles and possibly reassigns Gid values and Sender ID values in a group.

12.2.1.1. Recycling of Group Identifiers

Since the Gid value changes every time a group is rekeyed, it can happen that, after several rekeying instances, the whole space of Gid values has been used for the group in question. When this happens, the Group Manager has no available Gid values to use that have never been assigned to the group during the group's lifetime.

The occurrence of such an event and how long it would take to occur depend on the format and encoding of Gid values used in the group (see, e.g., Appendix C), as well as on the frequency of rekeying instances yielding a change of Gid value. Independently for each group under its control, the Group Manager can take one of the two following approaches:

- The Group Manager does not reassign Gid values. That is, once the whole space of Gid values has been used for a group, the Group Manager terminates the group and may re-establish a new group.
- While the Gid value changes every time a group is rekeyed, the Group Manager can reassign Gid values previously used during a group's lifetime. By doing so, the group can continue to exist even once the whole space of Gid values has been used.

  The Group Manager **MAY** support and use this approach, according to what is specified in Section 12.2.1.1.1.

12.2.1.1.1. Reassignment of Group Identifiers

If the Group Manager performs the reassignment of Gid values previously used during a group's lifetime, the Group Manager **MUST** take additional actions when handling Gid values and rekeying the group, as specified below.

When a node (re-)joins the group and it is provided with the current Gid to use in the group, the Group Manager considers such a Gid as the Birth Gid of that endpoint for that group. For each group member, the Group Manager **MUST** store the latest corresponding Birth Gid until that member leaves the group. If the endpoint has in fact re-joined the group, the newly determined Birth Gid overwrites the one currently stored.

When establishing a new Security Context for the group, the Group Manager takes the additional following step between Steps 1 and 2 of Section 12.2.

A. The Group Manager **MUST** check if the new Gid to be distributed is equal to the Birth Gid of any of the current group members. If any of such "elder members" is found in the group, then:

- The Group Manager **MUST** evict the elder members from the group. That is, the Group Manager **MUST** terminate their membership and, in the following steps, it **MUST** rekey the group in such a way that the new keying material is not provided to those evicted elder members.

  This ensures that any response from the same server to the request of a long exchange can never successfully match against the request of two different long exchanges.

  The excluded elder members could eventually re-join the group, thus terminating any of their ongoing long exchanges (see Section 5.1).

Therefore, it is ensured by construction that no client can have with the same server two ongoing long exchanges, such that the two respective requests were protected using the same Partial IV, Gid, and Sender ID.

12.2.1.2. Recycling of Sender IDs

From the moment a Gid is assigned to a group (T_start) until the moment when a new Gid is assigned to that same group, the following restrictions apply within the group:

- The Group Manager **MUST NOT** assign a Sender ID that was already the Sender ID of an endpoint in the group at T_start.
- The Group Manager **MUST NOT** assign a given Sender ID more than once.

That is, under the ongoing use of the current Gid, a given Sender ID is not reassigned to the same or a different endpoint. This prevents from reusing a Sender ID ('kid') with the same triple (Gid, Master Secret, Master Salt). Within these restrictions, the Group Manager can assign a Sender ID used under an old Gid value (including under a same, recycled Gid value), thus avoiding Sender ID values to irrecoverably grow in size.

Even when an endpoint joining a group is recognized as a current member of that group, e.g., through the ongoing secure communication association, the Group Manager **MUST** assign a new Sender ID different from the one currently used by the endpoint in the group unless the group is rekeyed first and a new Gid value is established.

12.2.1.3. Relation between Identifiers and Keying Material

Figure 4 gives an overview of the different identifiers and keying material components, considering their relation and possible reuse across group rekeying.

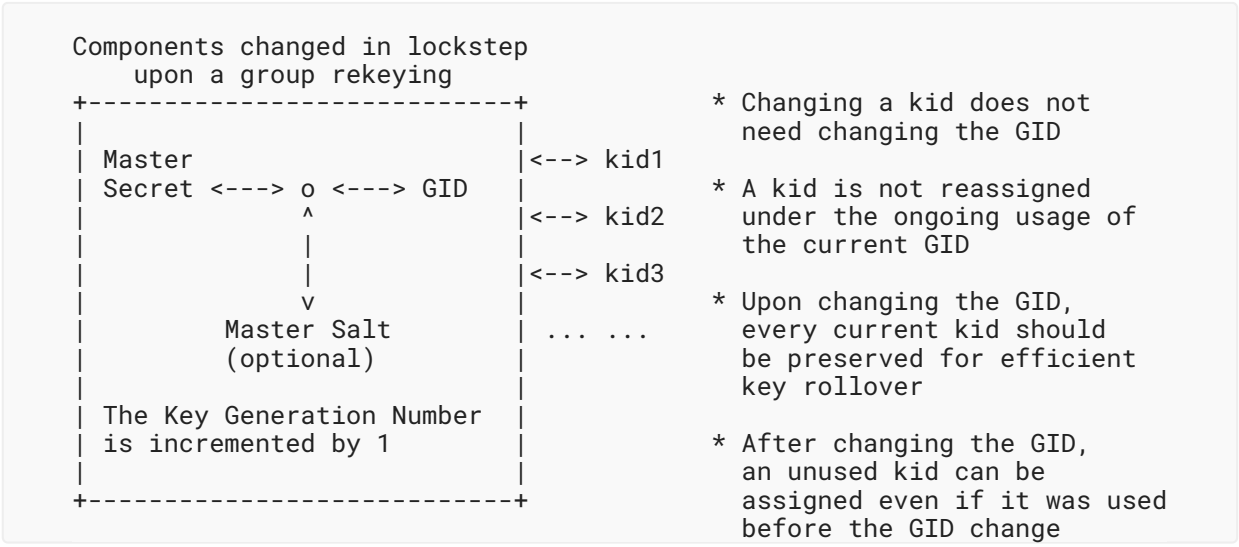


Figure 4: Relations Among Keying Material Components

12.3. Support for Signature Checkers

The Group Manager may serve signature checkers, e.g., intermediary gateways, which verify countersignatures of messages protected in group mode (see Section 7.5). These entities do not join a group as members, but they can retrieve authentication credentials of group members and other selected group data from the Group Manager.

In order to verify countersignatures of messages in a group, a signature checker needs to retrieve the following information about the group from the responsible Group Manager:

- The current ID Context (Gid) used in the group.
- The authentication credentials and other information (e.g., the Sender IDs) associated with the group members as well as the authentication credential of the Group Manager.

  If the signature checker is provided with a CWT or a CCS for a given entity, then the authentication credential associated with that entity is the untagged CWT or CCS.

  If the signature checker is provided with a chain or a bag of X.509/C509 certificates, or of CWTs, or of CCSs for a given entity, then the authentication credential associated with that entity is the end-entity certificate or end-entity untagged CWT/CCS.
- The current Signature Encryption Key used in the group (see Section 2.1.9).
- The identifiers of the algorithms used in the group (see Section 2), i.e., i) the HKDF Algorithm, ii) the Group Encryption Algorithm and Signature Algorithm, and iii) the AEAD Algorithm and Pairwise Key Agreement Algorithm, if such parameters are set in the Common Context (see Sections 2.1.1 and 2.1.10).

A signature checker **MUST** be authorized before it can retrieve such information, e.g., with the use of [ACE-KEY-GROUP-OSCORE].

13. Security Considerations

The same considerations from Appendix D of [RFC8613] on the security properties of OSCORE apply for Group OSCORE with the differences discussed in the following. Those considerations are about the reference threat model, the support for proxy operations, the protection of message fields, and the uniqueness of (key, nonce) pairs (which is further discussed in Section 13.3 of this document).

For Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys, which are used to provide source authentication in group mode by means of countersignatures (see Section 13.1) and in pairwise mode by using Diffie-Hellman (see Section 13.2). The key pair can, for example, be generated by the endpoint or provisioned during manufacturing.

Note that, even if an endpoint is authorized to be a group member and to take part in group communications, there is a risk that it behaves inappropriately. For instance, it can forward the content of messages in the group to unauthorized entities. However, in many use cases, the

devices in the group belong to a common authority and are configured by a commissioner (see Appendix B), which limits this risk in practice and enables a prompt detection/reaction in the case of misbehaving.

With respect to unprotected message fields, the following applies. First, the 'kid context' of request messages is part of the AAD, making it possible to keep long exchanges active safely beyond a possible change of ID Context (Gid) following a group rekeying (see Section 3.4). Second, the countersignature included in a Group OSCORE message protected in group mode is also computed over the value of the OSCORE Option, which is also part of the AAD used in the signing process. This is further discussed in Section 13.7 of this document.

In accordance with [RFC8613], all elements used in Group OSCORE as opaque binary values (e.g., Sender ID, ID Context) are not to be interpreted as text, Unicode, or otherwise. Implementations ought not to apply encoding transformations to the content of those elements, e.g., UTF-8 decoding [RFC3629] or normalization. Information elements that may contain text such as those found within authentication credentials (e.g., X.509 distinguished names, CWT claims, or JSON Web Key fields) are to be treated as opaque structured data and to be interpreted only according to the rules of the credential format as defined in their respective specifications. This avoids misinterpretation, Unicode normalization attacks, or mismatches in identity comparison.

As discussed in Section 6.2.3 of [RFC10020], Group OSCORE addresses security attacks against CoAP as listed in Sections 11.2 through 11.6 of [RFC7252], especially when run over IP multicast.

Group OSCORE does not aim to meet the following properties:

- Verification of server aliveness, as discussed in Section 5.2.
- Protection of network addressing information, as discussed in Section 13.9.
- Management of group membership and group keying material, which is entrusted to the Group Manager (see Section 12). Related security considerations are discussed in Sections 13.4 and 13.5.
- Confidentiality protection of the OSCORE Option. Related privacy considerations are discussed in Section 13.18.

The rest of this section first discusses security aspects to be taken into account when using Group OSCORE. Then, it goes through aspects covered in the security considerations of OSCORE (see Section 12 of [RFC8613]) and discusses how they apply when Group OSCORE is used.

13.1. Security of the Group Mode

The group mode defined in Section 7 relies on shared group keying material to protect communication within a group. Using the group mode has the implications discussed below. This section uses the term 'group members' to describe endpoints that possess the latest version of the group keying material.

- Source authentication of messages sent to a group is ensured through a countersignature, which is computed by the sender endpoint using its own private key and then appended to the message payload. The countersignature is also encrypted using a keystream derived

from the group keying material (see Sections 4.1 and 4.2). This ensures group privacy, i.e., an attacker cannot track an endpoint over two groups by linking messages between the two groups unless the attacker is also a member of both groups.

- Messages are encrypted at a group level (group-level data confidentiality), i.e., they can be decrypted by any member of the group, but not by an external adversary or other external entities other than the Group Manager responsible for the group.
- If the used Group Encryption Algorithm provides integrity protection, then it also ensures group authentication and proof of group membership, but not source authentication. That is, it ensures that a message sent to a group has been sent by a member of that group, but not necessarily by the alleged sender. In fact, any group member is able to derive the Sender Key used by the actual sender endpoint, and thus can compute a valid authentication tag. Therefore, the message content could originate from any of the current group members.

  Furthermore, if the used Group Encryption Algorithm does not provide integrity protection, then it does not ensure any level of message authentication or proof of group membership.

  On the other hand, proof of group membership is always ensured by construction through the strict management of the group keying material (see Section 12.2). That is, the group is rekeyed when members leave and the current group members are informed of former group members. Thus, a current group member storing the latest group keying material does not store the authentication credential of any former group member.

  This allows a recipient endpoint to rely on the stored authentication credentials and public keys included therein in order to always confidently verify the group membership of a sender endpoint when processing an incoming message, i.e., to verify that the sender endpoint was a group member when it signed the message. In turn, this prevents a former group member from possibly re-signing and injecting in the group a stored message that was protected with old keying material. A case in point is discussed in Section 13.1.1.

The security properties of the group mode are summarized below.

1. Asymmetric source authentication by means of a countersignature.
2. Symmetric group authentication by means of an authentication tag (only for Group Encryption Algorithms providing integrity protection).
3. Symmetric group confidentiality by means of symmetric encryption.
4. Proof of group membership by strictly managing the group keying material as well as by means of integrity tags when using a Group Encryption Algorithm that provides also integrity protection.
5. Group privacy by encrypting the countersignature.

The group mode fulfills the security properties above while also displaying the following benefits. First, the use of a Group Encryption Algorithm that does not provide integrity protection results in a minimal communication overhead by limiting the message payload to the ciphertext without integrity tag together with the encrypted countersignature. Second, it is possible to deploy semi-trusted entities such as signature checkers (see Section 12.3), which can break property 5, but cannot break properties 1, 2, 3, and 4.

13.1.1. Example of Need for Proof of Group Membership

As a case in point, the importance of reliable proof of group membership is evident in a group where the Group Encryption Algorithm does not provide integrity protection. Suppose that a group member leaves the group and, after the group rekeying, associates with the group as external signature checker (see Section 7.5). When doing so, it obtains the new Signature Encryption Key from the Group Manager, from which it can derive keystreams for encrypting and decrypting the countersignatures of messages protected in group mode.

However, when participating in the group rekeying, the current group members deleted the Recipient Context and authentication credential of the former group member. Consequently, the signature checker is not able to successfully inject messages protected in group mode, as encrypted with the old group keying material, signed with its own private key, and with the countersignature encrypted by means of the latest Signature Encryption Key. That is, if the signature checker attempts to do that, then the group members will fail to verify the messages from the signature checker and thus will discard those messages.

13.2. Security of the Pairwise Mode

The pairwise mode defined in Section 8 protects messages by using pairwise symmetric keys, derived from the static-static Diffie-Hellman shared secret computed from the asymmetric keys of the sender and recipient endpoint (see Section 2.5).

The used AEAD Algorithm **MUST** provide integrity protection. Therefore, the pairwise mode ensures both pairwise data confidentiality and source authentication of messages without using countersignatures. Furthermore, the recipient endpoint achieves proof of group membership for the sender endpoint since only current group members have the required keying material to derive a valid Pairwise Sender/Recipient Key.

Finally, the pairwise mode ensures group privacy, i.e., an attacker cannot track an endpoint over two groups by linking messages between the two groups unless the attacker is also a member of both groups. This follows from two different groups using different and uncorrelated group keying material, which yields different and uncorrelated pairwise keys for the same endpoint in any two groups. Therefore, the authentication tags generated by an endpoint in a group have no correlation with those generated by the same endpoint in another group.

The security properties of the pairwise mode are summarized below.

1. Symmetric source authentication by means of an authentication tag.
2. Symmetric pairwise confidentiality by means of symmetric encryption.
3. Proof of group membership by strictly managing the group keying material, as well as by means of integrity tags.
4. Group privacy by virtue of the uncorrelated pairwise keys used in any two different groups.

The long-term storing of the Diffie-Hellman shared secret is a potential security issue. In fact, if the shared secret of two group members is leaked, a third group member can exploit it to derive their pairwise keys and use those to impersonate either of the two group members to the other, or to decrypt previously stored messages exchanged between those two members and protected with their pairwise keys. The possibility of such leakage should be considered more likely than the leakage of a private key, which could be rather protected at a significantly higher level than generic memory, e.g., by using a Trusted Platform Module. Therefore, there is a trade-off between the maximum amount of time a same shared secret is stored and the frequency of its re-computing.

13.3. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [RFC8613] is also valid in group communication scenarios. That is, given an OSCORE group:

- Uniqueness of Sender IDs within the group is enforced by the Group Manager. In fact, from the moment when a Gid is assigned to a group until the moment when a new Gid is assigned to that same group, the Group Manager does not reassign a Sender ID within the group (see Section 12.2.1.2).
- Case A in Appendix D.4 of [RFC8613] concerns all requests as well as all responses including a Partial IV (e.g., Observe notifications [RFC7641] or any other subsequent responses after the first one). In this case, the same considerations from [RFC8613] apply here as well.
- Case B in Appendix D.4 of [RFC8613] concerns responses not including a Partial IV (e.g., a single response to a request). In this case, the same considerations from [RFC8613] apply here as well.

As a consequence, each message encrypted/decrypted with the same key is processed by using a different (ID_PIV, PIV) pair. This means that nonces used by any fixed encrypting endpoint are unique. Thus, each message is processed with a different (key, nonce) pair.

13.4. Management of Group Keying Material

The protocol described in this document should take into account the risk of compromise of group members. In particular, this document specifies that a key management scheme for secure revocation and renewal of Security Contexts and group keying material **MUST** be adopted.

[ACE-KEY-GROUP-OSCORE] specifies a simple rekeying scheme for renewing the Security Context in a group.

Alternative rekeying schemes that are more scalable with the group size may be needed in dynamic, large groups where endpoints can join and leave at any time in order to limit the impact on performance due to the Security Context and keying material update.

13.4.1. Denial of Service

An adversary may leverage the loss of Sender Contexts and/or Recipient Contexts as described in Section 2.6.1.1 in order to perform a Denial of Service attack and prevent some group members from communicating altogether.

That is, the adversary can first block the communication path between the Group Manager and some individual group members. This can be achieved, for instance, by injecting fake responses to DNS queries for the Group Manager hostname or by removing a network link used for routing traffic towards the Group Manager.

Then, the adversary can induce an unprepared reboot for some endpoints in the group, e.g., by triggering a short power outage. After that, such endpoints that have lost their Sender Context and/or Recipient Contexts following the reboot would not be able to obtain new Security Context parameters from the Group Manager as specified in Section 2.6.1.1. Thus, they would not be able to further communicate in the group until connectivity with the Group Manager is restored.

13.5. Update of Security Context and Key Rotation

A group member can receive a message shortly after the group has been rekeyed and new security parameters and keying material have been distributed by the Group Manager.

This may result in a client using an old Security Context to protect a request and a server using a different new Security Context to protect a corresponding response. As a consequence, clients may receive a response protected with a Security Context different from the one used to protect the corresponding request.

In particular, a server may first receive a request protected with the old Security Context, then install the new Security Context, and only after that produce a response to send back to the client. In such a case, as specified in Section 7.3, the server **MUST** protect the potential response using the new Security Context. Specifically, the server **MUST** include its Sender Sequence Number as Partial IV in the response and use it to build the nonce to protect the response. This prevents the nonce from the request from being reused with the new Security Context.

The client will process that response using the new Security Context, provided that it has installed the new security parameters and keying material before the message processing.

If block-wise transfer [RFC7959] is used, the same considerations from Section 10.3 of [RFC9594] apply.

Furthermore, as described below, a group rekeying may temporarily result in misaligned Security Contexts between the sender and recipient of a given message.

13.5.1. Late Update on the Sender

In this case, the sender protects a message using the old Security Context, i.e., before having installed the new Security Context. However, the recipient receives the message after having installed the new Security Context and is thus unable to correctly process it.

A possible way to mitigate this issue is to preserve the old retained Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context.

This makes particular sense when the recipient is a client, as it enables the client to process incoming responses protected with the old retained Security Context that was used to protect the associated request. If, as typically expected, the old Gid is not included in the response, the client will first fail to process the response using the latest Security Context and then use the old retained Security Context as a second attempt.

Instead, a recipient server can immediately process an incoming request with the old retained Security Context, as signaled by the old Gid that is always included in requests. However, the server could simply discard such an incoming request, which is preferable from a security point of view.

This tolerance preserves the processing of secure messages throughout a long-lasting key rotation, as group rekeying processes may likely take a long time to complete, especially in large groups. On the other hand, a former (compromised) group member can abusively take advantage of this and send messages protected with the old retained Security Context. Therefore, a conservative application policy should not permit the retention of old Security Contexts.

13.5.2. Late Update on the Recipient

In this case, the sender protects a message using the new Security Context, but the recipient receives that message before having installed the new Security Context. Therefore, the recipient would not be able to correctly process the message and hence discards it.

If the recipient installs the new Security Context shortly after that and the sender endpoint retransmits the message, the former will still be able to receive and correctly process the message.

In any case, the recipient should actively ask the Group Manager for an updated Security Context according to an application-defined policy, e.g., after a given number of unsuccessfully decrypted incoming messages.

13.6. Collision of Group Identifiers

If endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide.

This does not impair the security of the AEAD Algorithm and of the Group Encryption Algorithm. In fact, as long as the Master Secret is different for different groups and this condition holds over time, keys are different among different groups.

If multiple groups use the same IP multicast address, the entity assigning that address may help limiting the chances to experience such collisions of Group Identifiers. In particular, it may allow the Group Managers of those groups using the same IP multicast address to share their respective list of assigned Group Identifiers currently in use.

13.7. Cross-Group Message Injection

A given endpoint is allowed to and would likely use the same pair (private key, authentication credential) in multiple OSCORE groups, possibly administered by different Group Managers.

When a sender endpoint sends a message protected in pairwise mode to a recipient endpoint in an OSCORE group, a malicious group member may attempt to inject the message to a different OSCORE group also including the same endpoints (see Section 13.7.1).

This relies on altering the content of the OSCORE Option in such a way that the MAC in the ciphertext will still be correctly validated, which has a success probability depending on the size of the MAC.

As discussed in Section 13.7.2, the attack is practically infeasible if the message is protected in group mode, thanks to the countersignature also bound to the OSCORE Option through the AAD used in the signing process (see Section 3.4).

13.7.1. Attack Description

Consider the following:

- OSCORE groups G1 and G2, with ID Context (Gid) GID1 and GID2, respectively. Both G1 and G2 use the AEAD algorithm AES-CCM-16-64-128, i.e., the MAC of the ciphertext is 8 bytes in size.
- A sender endpoint X, which is a member of both G1 and G2 and uses the same pair (private key, authentication credential) in both groups. The endpoint X has Sender ID SID1 in G1 and Sender ID SID2 in G2. The pairs (SID1, GID1) and (SID2, GID2) identify the same authentication credential of X in G1 and G2, respectively.
- A recipient endpoint Y, which is a member of both G1 and G2 and uses the same pair (private key, authentication credential) in both groups. The endpoint Y has Sender ID SID3 in G1 and Sender ID SID4 in G2. The pairs (SID3, GID1) and (SID4, GID2) identify the same authentication credential of Y in G1 and G2, respectively.
- A malicious endpoint Z, which is also a member of both G1 and G2. Hence, Z is able to derive the Sender Keys used by X in G1 and G2.

When X sends a message M1 addressed to Y in G1 and protected in pairwise mode, Z can intercept M1 and attempt to forge a valid message M2 to be injected into G2, making it appear to have been sent by X to Y and therefore accepted as valid.

In more detail, Z intercepts and stops message M1 and forges a message M2 by changing the value of the OSCORE Option from M1 as follows: the 'kid context' is set to GID2 (rather than GID1), and the 'kid' is set to SID2 (rather than SID1). Then, Z injects message M2 as addressed to Y in G2.

Upon receiving M2, there is a probability equal to 2^{-64} that Y successfully verifies the same unchanged MAC by using the Pairwise Recipient Key associated with X in G2.

Note that Z does not know the pairwise keys of X and Y since it does not know and is not able to compute their shared Diffie-Hellman secret. Therefore, Z is not able to check offline if a performed forgery is actually valid, before sending the forged message to G2.

13.7.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected in group mode, the countersignature is also computed over the value of the OSCORE Option, which is part of the AAD used in the signing process (see Section 3.4).

That is, in addition to the ciphertext, the countersignature is computed over the ID Context (Gid) and the Partial IV (both of which are always present in requests), as well as the Sender ID of the message originator (which is always present in requests and in responses to requests protected in group mode).

Since the signing process also takes the ciphertext of the COSE_Encrypt0 object as input, the countersignature is bound not only to the intended OSCORE group (hence to the triplet composed of Master Secret, Master Salt, and ID Context), but also to a specific Sender ID in that group and the specific AEAD key derived from it (hence to the quartet composed of Master Secret, Master Salt, ID Context, and Sender ID).

This makes it practically infeasible to perform the attack described in Section 13.7.1 since it would require the adversary to additionally forge a valid countersignature that replaces the original one in the forged message M2.

13.8. Prevention of Group Cloning Attack

When using the group mode or the pairwise mode, the message protection covers also the Group Manager's authentication credential. This is included in the AAD used in the signing process and/or in the integrity-protected encryption process (see Section 3.4).

By doing so, an endpoint X that is a member of a group G1 cannot perform the following attack.

1. X sets up a group G2 where it acts as Group Manager.
2. X makes G2 a "clone" of G1, i.e., G1 and G2 use the same algorithms and have the same Master Secret, Master Salt, and ID Context.
3. X collects a message M sent to G1 and injects it in G2.
4. Members of G2 accept M and believe it to be originated in G2.

The attack above is effectively prevented since message M is protected by including the authentication credential of G1's Group Manager in the AAD. Therefore, members of G2 do not successfully verify and decrypt M since they correctly use the authentication credential of X as Group Manager of G2 when attempting to.

13.9. Group OSCORE for Unicast Requests

If a request is intended to be sent over unicast as addressed to a single group member, it is **NOT RECOMMENDED** for the client to protect the request by using the group mode as defined in Section 7.1.

This does not include the case where the client sends a request over unicast to a proxy to be forwarded to multiple intended recipients over multicast [RFC10020]. In this case, the client typically protects the request with the group mode, even though it is sent to the proxy over unicast (see Section 7).

If the client uses the group mode with its own Sender Key to protect a unicast request to a group member, an on-path adversary can, right then or later on, redirect that request to one/many different group member(s) over unicast, or to the whole OSCORE group over multicast. By doing so, the adversary can induce the target group member(s) to perform actions intended for one group member only. Note that the adversary can be external, i.e., the adversary does not need to also be a member of the OSCORE group.

This is due to the fact that the client is not able to indicate the single intended recipient in a way that is secure and possible to process for Group OSCORE on the server side. In particular, Group OSCORE does not protect network addressing information such as the IP address of the intended recipient server. It follows that the server(s) receiving the redirected request cannot verify whether that was the original intention of the client, and would thus simply assume so.

The impact of such an attack depends especially on the REST method of the request, i.e., the Inner CoAP Code of the protected request message. In particular, safe methods such as GET and FETCH would trigger (several) unintended responses from the targeted server(s), while not resulting in destructive behavior. On the other hand, non-safe methods such as PUT, POST, and PATCH/iPATCH would result in the target server(s) taking active actions on their resources and possible cyber-physical environment with the risk of destructive consequences and possible implications for safety.

Instead, a client can use the pairwise mode as defined in Section 8.3 in order to protect a request sent to a single group member by using pairwise keying material (see Section 2.5). This prevents the attack discussed above by construction, as only the intended server is able to derive the pairwise keying material used by the client to protect the request.

Before delivering to the application an incoming request protected in group mode that was sent over unicast, a server should carefully consider the impact that processing the request would have without dismissing the possibility that the request reception was in fact the result of the attack discussed above. This is especially important for endpoints implementing only a silent server since they do not support the pairwise mode and thus are able to process only requests protected in group mode. Application policies can also define specific exceptional cases where it is safe for a server to deliver such requests to the application, which can then take a final decision about whether acting on the request or not. For instance, such decision can leverage an application-level unique identifier of the server specified in the payload of the request.

In a group where the AEAD Algorithm and Pairwise Key Agreement Algorithm are set in the Security Context, an endpoint supporting the pairwise mode **SHOULD** use it to protect requests sent to a single group member over unicast. Using the group mode to protect such requests would make the request possible to decrypt and verify for any other group member; this may sometimes be a desired property (e.g., see [CoAP-MULT-NOTIF]).

The use of block-wise transfers [RFC7959] with group communication for CoAP is as discussed in Section 3.8 of [RFC10020]. Note that, after the first request targeting all servers in the group over multicast, a client can switch to sending unicast requests for retrieving further blocks from the servers. In particular, after a first block-wise request that targets all servers in the group and includes the CoAP Block2 Option, the retrieval of further blocks can use unicast requests, which should therefore be protected using the pairwise mode. Similarly, if the first request that targeted all servers in the group did not include the CoAP Block2 Option and the corresponding responses included the Block2 Option at the servers' own initiative, then the retrieval of further blocks can use unicast requests, which should therefore be protected using the pairwise mode.

Additional considerations are discussed in Section 9 with respect to requests including a CoAP Echo Option [RFC9175] that have to be sent over unicast, as employed in a challenge-response method for servers to achieve freshness or to initialize as valid a previously invalid Replay Window.

13.10. End-to-End Protection

The same considerations from Section 12.1 of [RFC8613] apply for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. However, it is not possible to combine (D)TLS and Group OSCORE for protecting message exchanges where messages are sent over multicast.

13.11. Master Secret

Group OSCORE derives the Security Context using the same construction used by OSCORE and by using the Group Identifier of a group as the related ID Context. Hence, the same required properties of the Security Context parameters discussed in Section 3.3 of [RFC8613] apply for this document.

With particular reference to the OSCORE Master Secret, it has to be kept secret among the members of the respective OSCORE group and the Group Manager responsible for that group. Also, the Master Secret must have a good amount of randomness, and the Group Manager can generate it offline using a good random number generator. This includes the case where the Group Manager rekeys the group by generating and distributing a new Master Secret.

13.12. Replay Protection

As in OSCORE [RFC8613], Group OSCORE relies on Sender Sequence Numbers included in the COSE message field 'Partial IV' and used to build nonces.

Note that the Partial IV of an endpoint does not necessarily grow monotonically. For instance, upon exhaustion of the endpoint's Sender Sequence Number space, the endpoint's Partial IV space also gets exhausted. As discussed in Section 2.6.3, this results in either the endpoint being individually rekeyed and getting a new Sender ID or the establishment of a new Security Context in the group. Therefore, uniqueness of (key, nonce) pairs (see Section 13.3) is preserved when a new Security Context is established.

Since one-to-many communication such as multicast usually involves unreliable transports, the simplification of the Replay Window to a size of 1 suggested in Section 7.4 of [RFC8613] is not viable with Group OSCORE unless exchanges in the group rely only on unicast messages.

A server's Replay Window may be initialized as invalid (see Section 2.6.1). The server can either retrieve a new Group OSCORE Security Context or make a Replay Window valid (see Section 9) before accepting further incoming messages from other group members.

13.13. Message Ordering

Assuming that the other endpoint is honest, Group OSCORE provides relative ordering of received messages. For a given Group OSCORE Security Context, the received Partial IV (when included) allows the recipient endpoint to determine the order in which requests or responses were sent by the other endpoint.

If the Partial IV was omitted in a response, this indicates that it was the oldest response from the sender endpoint to the corresponding request (like notification responses in OSCORE, see Section 7.4.1 of [RFC8613]). A received response is not older than the corresponding request.

13.14. Message Freshness

As in OSCORE, Group OSCORE provides only the guarantee that the request is not older than the Group OSCORE Security Context used to protect it. Other aspects of freshness are discussed in Section 5.2.

The challenge-response approach described in Section 9 provides an assurance of freshness of the request without depending on the honesty of the client. However, it can result in an impact on performance, which is undesirable or unbearable especially in large groups where many endpoints at the same time might join as new members.

Endpoints configured as silent servers are not able to perform the challenge-response described above, as they do not store a Sender Context to protect the 4.01 (Unauthorized) response to the client. Thus, silent servers should adopt alternative approaches to make their Replay Windows valid. For example, a silent server can retrieve or wait for new Security Context parameters from the Group Manager and derive new Recipient Contexts. When doing so, the Replay Windows of all Recipient Contexts become valid if they are not already. In particular, any invalid Replay Window is re-initialized as valid and with 0 as its current lower limit.

Since requests including the Echo Option are sent over unicast, a server can be the victim of the attack discussed in Section 13.9 if such requests are protected in group mode. Instead, protecting those requests with the pairwise mode prevents the attack above. In fact, only the server involved in the challenge-response exchange is able to derive the pairwise key used by the client to protect the request including the Echo Option.

In either case, an internal on-path adversary would not be able to transpose the Echo Option value of two different unicast requests sent by the same client to any two different servers in the group. In fact, even if the group mode was used, this would require the adversary to forge the

countersignature of both requests. As a consequence, each of the two servers remains able to selectively accept a request with the Echo Option only if it is waiting for that exact integrity-protected Echo Option value, and is thus the intended recipient.

13.15. Client Aliveness

Like in OSCORE (see Section 12.5 of [RFC8613]), a server may verify the aliveness of the client by using the CoAP Echo Option [RFC9175] as described in Section 9.

In the interest of avoiding otherwise unnecessary uses of such an approach, the server can exploit the fact that the received request cannot be older than the Security Context used to protect it. This effectively allows the server to verify the client aliveness relative to the installation of the latest group keying material.

13.16. Cryptographic Considerations

The same considerations from Section 12.6 of [RFC8613] about the maximum Sender Sequence Number apply for Group OSCORE.

As discussed in Section 2.6.2, an endpoint that experiences an exhaustion of its own Sender Sequence Number space **MUST NOT** protect further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint from reusing the same nonce with the same encryption key.

In order to renew its own Sender Context, the endpoint **SHOULD** inform the Group Manager, which can either renew the whole Security Context by means of group rekeying or provide only that endpoint with a new Sender ID value. In either case, the endpoint derives a new Sender Context, particularly a new Sender Key.

Additionally, the same considerations from Section 12.6 of [RFC8613] apply for Group OSCORE about building the nonce and the secrecy of the Security Context parameters.

The group mode uses the "encrypt-then-sign" construction, i.e., the countersignature is computed over the COSE_Encrypt0 object (see Section 3.1). This is motivated by enabling signature checkers (see Section 12.3), which do not join a group as members but are allowed to verify countersignatures of messages protected in group mode without being able to decrypt those messages (see Section 7.5).

If the Group Encryption Algorithm used in group mode provides integrity protection, countersignatures of COSE_Encrypt0 with short authentication tags do not provide the security properties associated with the same algorithm used in COSE_Sign (see Section 6 of [RFC9338]). To provide 128-bit security against collision attacks, the tag length **MUST** be at least 256 bits. A countersignature of a COSE_Encrypt0 with AES-CCM-16-64-128 provides at most 32 bits of integrity protection.

The derivation of pairwise keys defined in Section 2.5.1 is compatible with ECDSA and EdDSA asymmetric keys, but is not compatible with RSA asymmetric keys.

For the public key translation from Ed25519 (Ed448) to X25519 (X448) specified in Section 2.5.1, variable time methods can be used since the translation operates on public information. Any byte string of appropriate length is accepted as a public key for X25519 (X448) in [RFC7748]. It is therefore not necessary for security to validate the translated public key (assuming the translation was successful).

The security of using the same key pair for Diffie-Hellman and for signing (by considering the ECDH procedure in Section 2.5 as a Key Encapsulation Mechanism (KEM)) is demonstrated in [Degabriele] and [Thormarker].

Except when using X25519 and X448, applications that use the ECDH-based KEM in Section 2.5 are assumed to verify that a peer endpoint's public key is on the expected curve and that the shared secret is not the point at infinity. The KEM in [Degabriele] checks that the shared secret is different from the point at infinity, as does the procedure in Section 5.7.1.2 of [NIST-800-56A], which is referenced in Section 2.5.1.

By extending Theorem 2 of [Degabriele], [Thormarker] shows that the same key pair can be used with X25519 and Ed25519 (X448 and Ed448) for the KEM specified in Section 2.5. By symmetry in the KEM used in this document, both endpoints can consider themselves to have the recipient role in the KEM -- as discussed in Section 7 of [Thormarker] -- and rely on the mentioned proofs for the security of their key pairs.

Theorem 3 in [Degabriele] shows that the same key pair can be used for an ECDH-based KEM and ECDSA. The KEM uses a different Key Derivation Function (KDF) than in Section 2.5, but the proof only depends on that the KDF has certain required properties, which are the typical assumptions about HKDF, e.g., that output keys are pseudorandom. In order to comply with the assumptions of Theorem 3, received public keys **MUST** be successfully validated; see Section 5.6.2.3.4 of [NIST-800-56A]. The validation **MAY** be performed by a trusted Group Manager. For [Degabriele] to apply as it is written, public keys need to be in the expected subgroup. For this, we rely on cofactor Diffie-Hellman as per Section 5.7.1.2 of [NIST-800-56A], which is referenced in Section 2.5.1.

HashEdDSA variants of Ed25519 and Ed448 are not used by COSE (see Section 2.2 of [RFC9053]) and are not covered by the analysis in [Thormarker]. Hence, they **MUST NOT** be used with the public keys used to derive pairwise keys as specified in this document.

13.17. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] apply for Group OSCORE.

13.18. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and of all the CoAP options that are not used for proxy operations. In particular, CoAP options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same privacy considerations from Section 12.8 of [RFC8613] apply for Group OSCORE, with the following addition.

- When protecting a message in group mode, the countersignature is encrypted by using a keystream derived from the group keying material (see Sections 4.1 and 4.2). This ensures group privacy. That is, an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless the attacker is also a member of those groups.

Furthermore, the following privacy considerations apply about the OSCORE Option, which may reveal information on the communicating endpoints.

- The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. When both a request and the corresponding responses include the 'kid' parameter, this may reveal information about both a client sending a request and all the possibly replying servers sending their own individual response.
- The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Security Context, reveals information about the sender endpoint. In particular, it reveals that the sender endpoint is a member of a particular OSCORE group, whose current GID is indicated in the 'kid context' parameter.

When receiving a group request, each of the recipient endpoints can reply with a response that includes its Sender ID as the value of the 'kid' parameter. All these responses will be matchable with the request through the CoAP Token. Thus, even if these responses do not include a 'kid context' parameter, it becomes possible to understand that the responder endpoints are in the same group of the requester endpoint.

Furthermore, using the approach described in Section 9 to make Replay Windows valid may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the Replay Window of each known client on a clean shutdown.

Finally, the approach described in Section 13.6 to prevent collisions of Group Identifiers from different Group Managers may reveal information about events in the respective OSCORE groups. In particular, a Group Identifier changes when the corresponding group is rekeyed. Thus, Group Managers might use the shared list of Group Identifiers to infer the rate and patterns of group membership changes triggering a group rekeying, e.g., due to newly joined members or evicted (compromised) members. In order to alleviate this privacy concern, it should be hidden from the Group Managers which exact Group Manager has currently assigned which Group Identifiers in its OSCORE groups.

14. IANA Considerations

IANA has performed the actions described below.

14.1. OSCORE Flag Bits Registry

IANA has added the following entry to the "OSCORE Flag Bits" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group:

Bit Position	Name	Description	Reference
2	Group Flag	For using a Group OSCORE Security Context, set to 1 if the message is protected with the group mode	RFC 10021

Table 1: OSCORE Flag Bits Registry

14.2. Target Attributes Registry

IANA has added the following entry to the "Target Attributes" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group:

Attribute Name	Brief Description	Change Controller	Reference
gosc	Hint: resource only accessible using Group OSCORE or OSCORE	IETF	RFC 10021, Section 11

Table 2: Target Attributes Registry

15. References

15.1. Normative References

[COSE.Algorithms] IANA, "COSE Algorithms", <https://www.iana.org/assignments/cose>.

[NIST-800-56A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", National Institute of Standards and Technology, NIST SP 800-56Ar3, DOI 10.6028/NIST.SP.800-56Ar3, April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/info/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/info/rfc9053>.

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/info/rfc9175>.

[RFC9338] Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <https://www.rfc-editor.org/info/rfc9338>.

[RFC9459] Housley, R. and H. Tschofenig, "CBOR Object Signing and Encryption (COSE): AES-CTR and AES-CBC", RFC 9459, DOI 10.17487/RFC9459, September 2023, <https://www.rfc-editor.org/info/rfc9459>.

[RFC10020] Dijk, E. and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 10020, DOI 10.17487/RFC10020, November 2026, <https://www.rfc-editor.org/info/rfc10020>.

15.2. Informative References

[ACE-KEY-GROUP-OSCORE] Tiloca, M. and F. Palombini, "Key Management for Group Object Security for Constrained RESTful Environments (Group OSCORE) Using Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-21, 14 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-21>.

[C509-CERT] Mattsson, J. P., Selander, G., Raza, S., Höglund, J., Furuhed, M., and L. Liao, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-21, 24 September 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-21>.

[CACHEABLE-OSCORE] Amsüss, C. and M. Tiloca, "End-to-End Protected and Cacheable CoAP Responses using Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-cacheable-oscore-02, 6 July 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-core-cacheable-oscore-02>.

[CoAP-GROUP-PROXY] Tiloca, M. and E. Dijk, "Proxy Operations in Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-proxy-07, 2 September 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-proxy-07>.

[CoAP-MULT-NOTIF] Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-17, 2 November 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-core-observe-multicast-notifications-17>.

[CoAP-SEC-PROT] Mattsson, J. P., Palombini, F., and M. Vučinić, "Comparison of CoAP Security Protocols", Work in Progress, Internet-Draft, draft-ietf-iotops-security-protocol-comparison-09, 4 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-security-protocol-comparison-09>.

[CURVE-REPR] Struik, R., "Alternative Elliptic Curve Representations", Work in Progress, Internet-Draft, draft-ietf-lwig-curve-representations-23, 21 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-lwig-curve-representations-23>.

[Degabriele] Degabriele, J. P., Lehmann, A., Paterson, K. G., Smart, N. P., and M. Strefler, "On the Joint Security of Encryption and Signature in EMV", Cryptology ePrint Archive, Paper 2011/615, December 2011, <https://eprint.iacr.org/2011/615>.

[OSCORE-PROXIES] Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-ietf-core-oscore-capable-proxies-08, 2 November 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-capable-proxies-08>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <https://www.rfc-editor.org/info/rfc4340>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <https://www.rfc-editor.org/info/rfc8075>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/info/rfc9002>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/info/rfc9200>.

[RFC9594] Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication Using Authentication and Authorization for Constrained Environments (ACE)", RFC 9594, DOI 10.17487/RFC9594, September 2024, <https://www.rfc-editor.org/info/rfc9594>.

[SIGS-WITH-NOISE] Mattsson, J. P., Thormarker, E., and S. Ruohomaa, "Hedged ECDSA and EdDSA Signatures", Work in Progress, Internet-Draft, draft-irtf-cfrg-det-sigs-with-noise-05, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-det-sigs-with-noise-05>.

[Thormarker] Thormarker, E., "On using the same key pair for Ed25519 and an X25519 based KEM", Cryptology ePrint Archive, Paper 2021/509, April 2021, <https://eprint.iacr.org/2021/509>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the protocol described in this document. The rest of this section refers to three types of groups:

- Application group, i.e., a set of CoAP endpoints that share a common pool of resources.
- Security group, as defined in Section 1.1 of this document. Between security groups and application groups, there can be a many-to-many, one-to-many, many-to-one, or one-to-one relationship.
- CoAP group, i.e., a set of CoAP endpoints where each endpoint is configured to receive one-to-many CoAP requests, e.g., sent to the group's associated IP multicast address and UDP port as defined in [RFC10020]. An endpoint may be a member of multiple CoAP groups. Between CoAP groups and application groups, there can be a many-to-many, one-to-many, many-to-one, or one-to-one relationship. Note that a device sending a CoAP request to a CoAP group is not necessarily a member of that group; it is a member only if it also has a CoAP server endpoint listening for requests sent to the group's associated IP multicast address and UDP port. In order to provide secure group communication, all members of a CoAP group as well as all further endpoints configured only as clients sending CoAP (multicast) requests to the CoAP group have to be members of a security group. Between CoAP groups and security groups, there can be a many-to-many, one-to-many, many-to-one, or one-to-one relationship.

A.1. Assumptions

The following points are assumed to be already addressed and are out of the scope of this document.

Multicast communication topology: This document considers both 1-to-N (one sender and multiple recipients) and M-to-N (multiple senders and multiple recipients) communication topologies. The 1-to-N communication topology is the simplest group communication scenario that would serve the needs of a typical Low-Power and Lossy Network (LLN). Examples of use cases that benefit from secure group communication are provided in Appendix B.

In a 1-to-N communication model, only a single client transmits data to the CoAP group in the form of request messages; in an M-to-N communication model (where M and N do not necessarily have the same value), M clients transmit data to the CoAP group. According to [RFC10020], any possible proxy entity is supposed to know about the clients. Also, every client expects and is able to handle multiple response messages associated with a same request sent to the CoAP group.

Group size: Security solutions for group communication should be able to adequately support different and possibly large security groups. The group size is the current number of members in a security group. In the use cases mentioned in this document, the number of clients (normally the controlling devices) is expected to be much smaller than the number of servers (i.e., the controlled devices). A security solution for group communication that

supports 1 to 50 clients would be able to properly cover the group sizes required for most use cases that are relevant for this document. The maximum group size is expected to be in the range of hundreds to thousands of devices, with large groups easier to manage if including several silent servers. Security groups larger than that should be divided into smaller independent groups. One should not assume that the set of members of a security group remains fixed. That is, the group membership is subject to changes, possibly on a frequent basis.

Communication with the Group Manager: An endpoint must use a secure dedicated channel when communicating with the Group Manager, also in the case that it is not registered as a member of the security group.

Provisioning and management of Security Contexts: A Security Context must be established among the members of the security group. A secure mechanism must be used to generate, revoke, and (re-)distribute keying material, communication policies, and security parameters in the security group. The actual provisioning and management of the Security Context is out of the scope of this document.

Multicast data security cipher suite: All members of a security group must use the same cipher suite to provide authenticity, integrity, and confidentiality of messages in the group. The cipher suite is specified as part of the Security Context.

Ensuring backward security: A new device joining the security group should not have access to any old Security Contexts used before its joining. This ensures that a new member of the security group is not able to decrypt confidential data sent before it has joined the security group. The adopted key management scheme should ensure that the Security Context is updated to ensure backward confidentiality. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a new member's joining has to be defined as part of the group key management scheme.

Ensuring forward security: Entities that leave the security group should not have access to any future Security Contexts or message exchanged within the security group after their leaving. This ensures that a former member of the security group is not able to decrypt confidential data sent within the security group anymore. Also, it ensures that a former member is not able to send protected messages to the security group anymore. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a member's leaving has to be defined as part of the group key management scheme.

A.2. Security Objectives

The protocol described in this document aims at fulfilling the following security objectives:

Data replay protection: Request messages or response messages replayed within the security group must be detected.

Data confidentiality: Messages sent within the security group shall be encrypted.

Group-level data confidentiality: The group mode provides group-level data confidentiality since messages are encrypted at a group level, i.e., in such a way that they can be decrypted by any member of the security group, but not by an external adversary or other external entities other than the Group Manager responsible for the group.

Pairwise data confidentiality: The pairwise mode especially provides pairwise data confidentiality since messages are encrypted using pairwise keying material shared between any two group members; hence, they can be decrypted only by the intended single recipient.

Source message authentication: Messages sent within the security group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the security group, and in particular by a specific, identifiable member of the security group.

Message integrity: Messages sent within the security group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with, either by a group member or by an external adversary or other external entities that are not members of the security group.

Message ordering: It must be possible to determine the ordering of messages coming from a single sender. Like in OSCORE [RFC8613], a recipient endpoint can determine the relative order of requests or responses from another sender endpoint by means of their Partial IV. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [RFC10020] provides the necessary background for multicast-based CoAP communication with particular reference to LLNs and resource-constrained environments. The interested reader is encouraged to first read [RFC10020] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication and refers to the three types of groups from Appendix A. Specific security requirements for these use cases are discussed in Appendix A.

Lighting control: Consider a building equipped with IP-connected lighting devices, switches, and border routers. The lighting devices acting as servers are organized into application groups and CoAP groups, according to their physical location in the building. For instance, lighting devices in a room or corridor can be configured as members of a single application group and corresponding CoAP group. Those lighting devices together with the switches acting as clients in the same room or corridor can be configured as members of the corresponding security group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in the CoAP group, while border routers connected to an IP network backbone (which is also multicast-enabled) can be used to interconnect routers in the building. Consequently, this would also enable logical groups to be formed even if devices with a role in the lighting application may be physically in different subnets (e.g., on wired and wireless networks). Connectivity between lighting devices may be realized, for instance, by means of IPv6 and (border) routers supporting 6LoWPAN [RFC4944] [RFC6282]. Group communication enables synchronous operation of a set of connected lights,

ensuring that the light preset (e.g., dimming level or color) of a large set of luminaires are changed at the same perceived time. This is especially useful for providing a visual synchronicity of light effects to the user. As a practical guideline, events within a 200 ms interval are perceived as simultaneous by humans, which is necessary to ensure in many setups. Devices may reply back to the switches that issue on/off/dimming commands in order to report about the execution of the requested operation (e.g., OK, failure, or error) and their current operational status. In a typical lighting control scenario, a single switch is the only entity responsible for sending commands to a set of lighting devices. In more advanced lighting control use cases, an M-to-N communication topology would be required, e.g., if multiple sensors (presence or daylight) are responsible for triggering events to a set of lighting devices. Especially in professional lighting scenarios, the roles of client and server are configured by the lighting commissioner, and devices strictly follow those roles.

Integrated building control: Enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation, and air-conditioning units to predefined presets. Controlled units can be organized into application groups and CoAP groups in order to reflect their physical position in the building, e.g., devices in the same room can be configured as members of a single application group and corresponding CoAP group. As a practical guideline, events within intervals of seconds are typically acceptable. Controlled units may reply back to the BACS issuing control commands in order to report about the execution of the requested operation (e.g., OK, failure, or error) and their current operational status.

Software and firmware updates: Software and firmware updates often comprise quite a large amount of data. This can overload an LLN that is otherwise typically used to dealing with only small amounts of data on an infrequent basis. Rather than sending software and firmware updates as unicast messages to each individual device, multicasting such updated data to a larger set of devices at once displays a number of benefits. For instance, it can significantly reduce the network load and decrease the overall time latency for propagating this data to all devices. Even if the complete update process itself is secured, securing the individual messages is important, if updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids that devices store large amounts of partially corrupted data and that they detect tampering hereof only after all data has been received. Devices receiving software and firmware updates may reply back in order to provide feedback about the execution of the update operation (e.g., OK, failure, or error) and their current operational status.

Parameter and configuration update: By means of multicast communication, it is possible to update the settings of a set of similar devices, both simultaneously and efficiently. Possible parameters are related, for instance, to network load management or network access control. Devices receiving parameter and configuration updates may reply back in order to provide feedback about the execution of the update operation (e.g., OK, failure, or error) and their current operational status.

Commissioning of LLN systems: A commissioning device is responsible for querying all devices in the local network or a selected subset of them in order to discover their presence and be aware of their capabilities, default configuration, and operating conditions. Queried devices displaying similarities in their capabilities and features or sharing a common physical

location can be configured as members of a single application group and corresponding CoAP group. Queried devices are expected to reply back to the commissioning device in order to notify their presence and provide the requested information and their current operational status.

Emergency multicast: A particular emergency-related information (e.g., natural disaster) is generated and multicast by an emergency notifier and relayed to multiple devices. The latter may reply back to the emergency notifier in order to provide their feedback and local information related to the ongoing emergency. This kind of setup should additionally rely on a fault-tolerant multicast algorithm, such as the Multicast Protocol for Low-Power and Lossy Networks (MPL).

Appendix C. Example of Group Identifier Format

This section provides an example of how the Group Identifier (Gid) can be formatted. That is, the Gid can be composed of two parts, namely a Group Prefix and a Group Epoch.

For each group, the Group Prefix is constant over time and is uniquely defined in the set of all the groups associated with the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application-specific. The size of the Group Prefix directly impacts the maximum number of distinct groups under the same Group Manager.

The Group Epoch is set to 0 upon the group's initialization, and is incremented by 1 each time new keying material, together with a new Gid, is distributed to the group in order to establish a new Security Context (see Section 12.2).

As an example, a 3-byte Gid can be composed of i) a 1-byte Group Prefix '0xb1' interpreted as a raw byte string, and ii) a 2-byte Group Epoch interpreted as an unsigned integer ranging from 0 to 65535. Then, after having established the Common Context 61532 times in the group, its Gid will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group with a Group Manager that does not reassign Gid values (see Section 12.2.1.1) limits the total number of rekeying instances. With a Group Manager that does reassign Gid values, it limits the maximum number of rekeying instances that a long exchange can persist through. In either case, the Group Epoch size needs to be chosen depending on the expected rate of rekeying instances.

As discussed in Section 13.6, if endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible that Group Identifiers of different groups coincide at some point in time. In this case, a recipient has to handle coinciding Group Identifiers and has to try using different Security Contexts to process an incoming message until the right one is found and the message is correctly verified. Therefore, it is favorable that Group Identifiers from different Group Managers have a size that results in a small probability of collision. How small this probability should be is up to system designers.

Appendix D. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

1. Creating and managing OSCORE groups. This includes the assignment of a Gid to every newly created group, ensuring uniqueness of Gids within the set of its OSCORE groups and (optionally) the secure recycling of Gids.
2. Defining policies for authorizing the joining of its OSCORE groups.
3. Handling the join process to add new endpoints as group members.
4. Establishing the Common Context part of the Security Context and providing it to an authorized group member during the join process together with the corresponding Sender ID.
5. Updating the Key Generation Number and the Gid of its OSCORE groups upon renewing the respective Security Context.
6. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process or to current group members upon request of renewal or re-joining. This includes ensuring that:
   - Each Sender ID is unique within each of the OSCORE groups;
   - Each Sender ID is not reassigned within the same group since the latest time when the current Gid value was assigned to the group. That is, the Sender ID is not reassigned even to a current group member re-joining the same group, without a rekeying happening first.
7. Defining communication policies for each of its OSCORE groups and signaling them to new endpoints during the join process.
8. Renewing the Security Context of an OSCORE group upon membership change by revoking and renewing common security parameters and keying material (rekeying).
9. Providing the management keying material that a new endpoint requires to participate in the rekeying process consistently with the key management scheme used in the group joined by the new endpoint.
10. Assisting a group member that has missed a group rekeying instance to understand which authentication credentials and Recipient Contexts to delete, as associated with former group members.
11. Acting as the key repository in order to handle the authentication credentials of the members of its OSCORE groups and providing such authentication credentials and information about the associated group members to other members of the same group upon request. The actual storage of authentication credentials may be entrusted to a separate secure storage device or service.
12. Validating that the format and parameters of authentication credentials of group members are consistent with the public key algorithm and related parameters used in the respective OSCORE group.

The Group Manager specified in [ACE-KEY-GROUP-OSCORE] provides this functionality.

Acknowledgements

Jiye Park contributed as a coauthor to earlier draft versions of this document.

The authors sincerely thank Christian Amsüss, Stefan Beck, Mike Bishop, Rolf Blom, Carsten Bormann, Mohamed Boucadair, Deb Cooley, Esko Dijk, Gorry Fairhurst, Patrik Fältström, Martin Gunnarsson, Klaus Hartke, Richard Kelsey, Paul Kyzivat, Joerg Ott, Dave Robin, Jim Schaad, Ludwig Seitz, Orie Steele, Peter van der Stok, Ketan Talaulikar, Erik Thormarker, Mališa Vučinić, and Paul Wouters for their feedback and comments.

The work on this document has been partly supported by the Sweden's Innovation Agency VINNOVA and the Celtic-Next projects CRITISEC and CYPRESS; the H2020 projects SIFIS-Home (Grant agreement 952652) and ARCADIAN-IoT (Grant agreement 101020259); the SSF project SEC4Factory under the grant RIT17-0032; and the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-164 40 Kista
Sweden
Email: marco.tiloca@ri.se

Göran Selander
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: francesca.palombini@ericsson.com

John Preuß Mattsson
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: john.mattsson@ericsson.com

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: rikard.hoglund@ri.se